6. EXAMPLE PROGRAMS

Included with the M-API software are some example programs. These programs serve two purposes:

1. They offer specific examples of how to use the M-API routines.
2. They give the user an opportunity to verify, after installation, that the M-API routines were built correctly and are functioning as expected.

The example programs can be built by typing: "make test" while in the examples directory. The example programs can be executed, and if necessary built, with the output piped to the file "example.output" by typing: "make run". Individual routines can be built by typing his or her: "make *name*". The user will have to edit the makefile to work as expected for their version on UNIX.

NOTE: *Name* must be replaced with the example program name (e. g., example1).

The following example programs create, write, and read MODIS HDF arrays.

6.1 Example 1: Creating a Floating Point Array in FORTRAN

This FORTRAN program demonstrates how to create a 32-bit floating point array. The HDF file “arrex1.hdf” is created using OPMFIL. An initialized array containing 1's is created with CRMAR and then written to the HDF file with PMAR. Once the file has been written the file is closed with a call to CPMFIL. CPMFIL is used since it is a new HDF file.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
CRMAR	Initilizes an array structure in a file.
PMAR	Writes a subarray into an array structure.
CPMFIL	Completes and closes a MODIS file.

6.1.1 Source Code Listing for Example 1

```
      PROGRAM example1
        IMPLICIT NONE
        INCLUDE 'mapi.inc'

C This example program demonstrates how to open a new MODIS HDF file,
C create a new data array, put the new data array into the file, and close
C the MODIS HDF file.
C

C  DATA ARRAY
      REAL DATF(32,64)

C  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(3)

C  DIMENSION ARRAY
      INTEGER DIMS(2)

C  Start indices (0-based) for writing array
      INTEGER STA(2)

C  rank and error code
      INTEGER RANK, IER

C  Number of handles
      INTEGER NUMHANDLES

C  Array and group names
```

```
      CHARACTER*20 FILNM, ARRNM, GRPNM

C  Data type
      CHARACTER*(DATATYPELENMAX) DTYPE

C mdHandles array of ECS metadata groups in MCF
      CHARACTER*20 MDHANDLES

C Names of Global atributes to store ECS metadata in
      CHARACTER*20 HDFATTNMS

C  Initialize values
      DATA DATF/2048*1/
      DATA STA/2*0/
      DATA DIMS/32,64/
      DATA RANK/2/
      DATA ARRNM/'DATAFLOAT'/, GRPNM/’  ’  /
      DATA FILNM/'arrex1.hdf'/
      DATA NUMHANDLES/0/
      DTYPE = R32

      print*,'*** Example1 ***'

C  Open file
      IER = OPMFIL(FILNM, CREATE_FILE, MODFIL)

      IF(IER.EQ.MAPIOK) THEN
      PRINT *,'Openning of Modis file was successful!'
      END IF

C  Create array
      PRINT *,'Creating a Data array!'
      IER = CRMAR(MODFIL, ARRNM, GRPNM, DTYPE, RANK, DIMS)

C  Write to the array (note that the entire array is being written, so
C  data dimensions are equal to array dimensions)
      IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Writing array to MODIS HDF file!'
           IER = PMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,DATF)
      END IF
       PRINT *,'Wrote array to MODIS HDF file!'

C New MODIS file so use CPMFIL to close the file
        IER = CPMFIL(MODFIL, MDHANDLES, HDFATTNMS, NUMHANDLES)

      IF(IER .EQ. MAPIOK) THEN
           PRINT *,'MODIS file was successfully closed!'
      END IF
        PRINT *,'example1 done'
        PRINT *,’  ’
        STOP
      END
C  End of example
```

6.2 Example 2: Creating a Floating Point Array in C

This program is similar to the FORTRAN program. The program demonstrates how to create a 32-bit floating point array. When the program executes, an HDF file named 'arrex2.hdf is created for writing. A MODIS group name is created using createMODISgroup. The group name written to the HDF file using addMODISgroup. A 64 by 32 array is created using createMODISarray. The array is initialized to floating point values ranging from 0 to 2048 and then written to the HDF file with putMODISarray. Once the file has been written the file is closed with a call to closeMODISfile.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access: r, w, a).
`createMODISgroup`	Creates a MODIS group name.
`createMODISarray`	Initilizes an array structure in a file.
`putMODISarray`	Writes a array into an HDF file.
`completeMODISfile`	Completes a new MODIS file.

6.2.1 Source Code Listing for Example 2

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mapi.h"
#include "PGS_MET.h"
#include "PGS_PC.h"

#define MCF_FILE 10250

/*
 *    This example program demonstrates how to open a new MODIS HDF file,
 *    create a new data array, put the new data array into the file,
 *    and close the MODIS HDF file.
 */

main()
   {
   MODFILE  *modfile;         /* Modis file pointer */
   float data[64][32];                  /* Data Array */
   long cksum = 0;                      /* array check sum  */
   long dims[2] = {64,32};          /* Array dimensions */
   long sta[2]  = {0,0};            /* Array start indices (0-based) */
   long rank    = 2;                /* Array rank */
   char dtype[] = R32;                  /* Array type */
   char arrnm[] = "DFLOAT";         /* Array name */
   char grpnm[] = "Fred's group";   /* Group name */
   char fname[] = "arrex2.hdf";         /* Modis file name */
   char faccess[] = "w";                /* Modis file access */
   int mapier;                          /* Error code */
   int i,j;                   /* counters */
   int ret_val;
   double time1 = 90000000.0;
   double time2 = 90000100.0;
   double glats[4] = {10., 20., 30., 40.};
   double glons[4] = {50., 60., 70., 80.};
   long int gseq[4] = {1, 2, 3, 4};
   double douval=0.0;
   float floval=0.0;
   int intval=0;
   char *attrval;

   PGSt_MET_all_handles     mdHandles;
   ECSattr_names_for_all_handles HDFattrnms;
   long Numhandles = 2;

   printf(" *** Example2 ***\n");

  strcpy (HDFattrnms[1], MECS_CORE);
  strcpy (HDFattrnms[2], MECS_PRODUCT);

  /* initialize the MCF file */
  ret_val = PGS_MET_Init(MCF_FILE, mdHandles);
  if (ret_val != PGS_S_SUCCESS)
    printf("error in PGS_MET_Init %d\n",ret_val);

  /* Add metadata */
    douval = 100.;
   ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_SIZE_OF_GRANULE, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    /*  Try to retrieve this value from the metadata */
    douval = 0;
   ret_val = PGS_MET_GetSetAttr(mdHandles[1], MCORE_SIZE_OF_GRANULE, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_GetSetAttr %d\n",ret_val);
    printf("MCORE_SIZE_OF_GRANULE %lf\n", douval);

   attrval = (char *) malloc(27);

   attrval = "MOD03";
   ret_val = PGS_MET_SetAttr(mdHandles[1], "SHORTNAME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);
   attrval = "MODIS Geolocation";
   ret_val = PGS_MET_SetAttr(mdHandles[1], "LONGNAME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

   attrval = "1996-01-01T00:00:00.000000Z";
   ret_val = PGS_MET_SetAttr(mdHandles[1],"RANGEBEGINNINGDATETIME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

   attrval = "1996-01-01T00:02:30.000000Z";
   ret_val = PGS_MET_SetAttr(mdHandles[1], "RANGEENDINGDATETIME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

   /* Try setting a value again to see if we can modify them */

   attrval = "1996-01-01T00:02:31.000000Z";
   ret_val = PGS_MET_SetAttr(mdHandles[1], "RANGEENDINGDATETIME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

   ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_GRING_POINT_LAT, glats);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

   ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_GRING_POINT_LON, glons);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    douval = 40;
    ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_NORTH_BOUND, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    douval = 10;
    ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_SOUTH_BOUND, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    douval = 50;
    ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_WEST_BOUND, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    douval = 80;
    ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_EAST_BOUND, &douval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "Input file 1";
    ret_val = PGS_MET_SetAttr(mdHandles[1], "INPUTPOINTER.1", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);
```

```
attrval = "Ancillary input file 1";
ret_val = PGS_MET_SetAttr(mdHandles[1], "ANCILLARYINPUTPOINTER.1",
                    &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

attrval = "Ancillary input file 2";
ret_val = PGS_MET_SetAttr(mdHandles[1], "ANCILLARYINPUTPOINTER.2",
                    &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

attrval = "MOD.AM1.sample.L1.95001.000000.95001";
ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_GRAN_POINTER, &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

attrval = "Processing history";
ret_val = PGS_MET_SetAttr(mdHandles[1], "PROCESSINGHISTORYPOINTER",
              &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

attrval = "passed";
ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_AUTO_QUALITY, &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

attrval = "none";
ret_val = PGS_MET_SetAttr(mdHandles[1], "QUALITYFLAGEXPLANATION",
              &attrval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

intval = 0;
ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_PERCENT_MISSING, &intval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

intval = 0;
ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_PERCENT_OUT, &intval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

intval = 0;
ret_val = PGS_MET_SetAttr(mdHandles[1], "QAPERCENTINTERPOLATEDDATA",
              &intval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);

intval = 40;
ret_val = PGS_MET_SetAttr(mdHandles[1], MCORE_ORBIT_NUM, &intval);
if (ret_val != PGS_S_SUCCESS)
  printf("error in PGS_MET_SetAttr %d\n",ret_val);
```

```
    attrval = "day";
    ret_val = PGS_MET_SetAttr(mdHandles[1], "OPERATIONMODE", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "MOD.AM1.sample.l1.95001.0000000.95001";
    ret_val = PGS_MET_SetAttr(mdHandles[1], "MODISPRODUCTFILENAME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "1995-01-01T00:00:00.000000Z";
    ret_val = PGS_MET_SetAttr(mdHandles[1], "PROCESSINGDATETIME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "MODIS parameters";
    ret_val = PGS_MET_SetAttr(mdHandles[1], "SPSOPARAMETERS", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    intval = 20;
    ret_val = PGS_MET_SetAttr(mdHandles[1], "GRANULENUMBER", &intval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "MODIS ATBD";
    ret_val = PGS_MET_SetAttr(mdHandles[2], "ALGORITHMPACKAGENAME", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

    attrval = "GSFC";
    ret_val = PGS_MET_SetAttr(mdHandles[2], "PROCESSINGCENTER", &attrval);
    if (ret_val != PGS_S_SUCCESS)
      printf("error in PGS_MET_SetAttr %d\n",ret_val);

     for ( i = 1; i <= Numhandles; i++){
        printf("mdHandles[%d] = %s\n",i,mdHandles[i]);
        printf("HDFattrnms[%d] = %s\n",i,HDFattrnms[i]);
     }

   /* open the MODIS file */
   modfile= openMODISfile(fname, faccess);
   if (modfile==NULL){
      fprintf (stderr, "Error openning %s exiting \n",fname);
      exit(-1);
   }else{
     printf(" File: %s opened, access mode %s\n",fname,faccess);
   }

   /* create a group for the data */
   mapier = createMODISgroup(modfile, grpnm, NULL);
   if (mapier == MFAIL){
      fprintf (stderr, "Error creating group, exiting\n");
      exit(-1);
   }else{
     printf(" Group created, Name: %s,\n", grpnm);
   }

   /* Create array */

   mapier = createMODISarray(modfile,arrnm,grpnm,dtype,rank,dims);
   if (mapier == MFAIL){
      fprintf (stderr, "Error creating array, exiting\n");
      exit(-1);
   }else{
     printf(" Array created, Name: %s,\n",arrnm);
   }

   /* Write to the array (note that the entire array is being
      written, so data dimensions are equal to array dimensions */

   for (i=0; i< dims[0]; i++){
     for (j=0; j< dims[1]; j++){
       data[i][j]= (i+j) + 1000.0 ;
       cksum = cksum + data[i][j];
     }
   }
   mapier = putMODISarray(modfile,arrnm,grpnm,sta,dims,data);
   if (mapier == MFAIL){
     fprintf (stderr, "Error writing array, exiting\n");
     exit(-1);
   }else{
     printf(" Array check sum: %d \n",cksum);
     printf(" Put the array in the file...\n");
   }

   /* Close the MODIS-HDF file */

   mapier = completeMODISfile(&modfile, mdHandles, HDFattrnms, Numhandles);
   if (mapier == MFAIL){
      fprintf (stderr, "Error closing file, exiting\n");
       exit(-1);
   }else{
     printf(" File closed successfully\n");
   }
   exit(0);
}
/* End of example */
```

(This page intentionally left blank)

6.3 Example 2a: Open an HDF File to Read and Print an Array

This program demonstrates how to open an existing HDF file for the purpose of reading an array and then printing the arrray contents. When calling the program, at the command line the HDF filename and the SDS name (array name) are entered. For example: example1 created an HDF file named arrex1.hdf. The array or SDS name was DATAFLOAT. So, if one wants to read this file, type the following commands: example2a arrex1.hdf DATAFLOAT.

To read the HDF file created by example2, the commands are example2a arrex2.hdf DFLOAT. The procedure for reading an array is to first get the information about the array (i. e. the rank, dimensions, and datatype). This is done using getMODISarray. An optional parameter which was set in this example to “Fred’s group” is the group name. If the user wants to restrict the retrieval to a specific data group, then the group name should be be defined . Otherwise, this value can be set to NULL. Once this information is retrieved, then the system allocates enough memory to hold the array. The array is read into memory using the getMODISarray routine. The HDF file is closed using closeMODISfile. The array contents are then printed out.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access: r, w, a).
`getMODISardims`	Retrieves info about a MODIS HDF array.
`MODISsizeof`	Determines size in bytes of an array type.
`getMODISarray`	Retrieves an array or subarray.
`closeMODISfile`	Closes a MODIS file.

6.3.1 Source Code Listing for Example 2a

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mapi.h"

/*
**  This example program demonstrates how to open a MODIS HDF file,
**  read a given 2-D data array from the file, and close the file.
*/

main(int argc, char **argv)
   {
   MODFILE  *modfile;   /* Modis file pointer */
   void  *data = NULL;  /* Data Array */
   void  *dp;     /* Data Array scanner */
   long dims[2];  /* Array dimensions */
   long sta[2] = {0,0}; /* Array start indices (0-based) */
   long rank = 2; /* Array rank */
   char dtype[DATATYPELENMAX] = "\0";   /* Array type */
   char grpnm[] = "Fred's group";   /* Group name */
   long ier;                           /* Error code */
   int i,j; /* counters */
   float chksum = 0;                   /* check sum of array */

   /* Check calling arguments */
   if (argc != 3)
      {
      printf ("usage : example2a HDF_file_name SDS_name\n");
      exit(-1);
      }

   printf(" *** Example2a ***\n");
   /* Open the MODIS-HDF file */
   modfile= openMODISfile(argv[1], "r");
   if (modfile==NULL){
       printf(" Could not open MODIS-HDF file: %s for reading\n", argv[1]);
       exit(1);
   }else{
     printf(" File: %s opened for reading only\n", argv[1]);
   }

   /* Get dimensional information about the array */
   ier = getMODISardims(modfile,argv[2],grpnm,dtype,&rank,dims);

   printf(" group name = %s\n", grpnm);
   if (ier == MFAIL){
     printf(" Could not get dimensional information\n");
   }else{
     printf(" Dimensional data retrieved\n");
   }

   if ((ier == MAPIOK) && (rank == 2)){
      data = malloc(dims[0] * dims[1] * MODISsizeof(dtype));

      /* Read the entire array into the data buffer */
      ier = getMODISarray(modfile,argv[2],grpnm,sta,dims,data);
   }

   if (ier == MFAIL){
     printf("Errors reading array\n");
     exit(-1);
   }

   printf(" dtype: %s\n",dtype);
   /* Calculate sum of array and print */
    dp = data;
    for (i=sta[0]; i< (dims[0]+sta[0]); i++)
      for (j=sta[1]; j<(dims[1]+sta[1]); j++){
        chksum +=  *((float*)dp);
      dp = (char *)dp + MODISsizeof(dtype);
      }
    printf(" Array check sum: %5.f \n", chksum);

   /* Close the MODIS-HDF file */
   ier = closeMODISfile(&modfile);
   if ((ier == MFAIL) || (data == NULL)){
      free(data);
      printf (" example2a aborting\n");
      exit(-1);
   }else{
     printf(" File closed successfully\n");
   }

   free(data);
   printf (" example2a done\n");
   exit(0);
   }
/* End of example */
```

6.4 Example 3: Create an Integer Array in FORTRAN

This FORTRAN program demonstrates how to create a 32-bit integer array and write it to a file using a loop. The last dimension of the array is then given a descriptive label. The HDF file 'arrex3.hdf' is created using OPMFIL. An initialized array containing 1's is created with CRMAR. The last array dimension is named using PMDMIN. The array is then written to the file by looping. Once the file has been written the file is closed with a call to CPMFIL. CPMFIL is used since it is a new file.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
CRMAR	Initializes an array structure in a file.
PMDMIN	Writes an array dimension name to a file.
PMAR	Writes a subarray into an array structure.
CPMFIL	Completes a new MODIS file.

6.4.1 Source Code Listing for Example 3

```
      PROGRAM example3
      IMPLICIT none
      INCLUDE 'mapi.inc'
        integer pgs_met_init
        integer pgs_met_setattr_d
        integer pgs_met_getsetattr_d

C  DATA ARRAY
      INTEGER IDATA(15,20)

C  Counter
      INTEGER I

C  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(MODFILLEN)

C  DIMENSION ARRAY
      INTEGER DIMS(3)

C  Start indices (0-based) for writing array
      INTEGER STA(3)

C  rank and error code
      INTEGER RANK, IER
```

```
C  Number of handles
        INTEGER NUMHANDLES

C  Array and group names
      CHARACTER*20 ARRNM, GRPNM, FILNM
C  Dimension name
      CHARACTER*20 DIMNM
C  Data type
      CHARACTER*(DATATYPELENMAX) DTYPE, ATYPE
C  Array Data type, array attribute
      CHARACTER*20 ATTR
C  Attribute value
      CHARACTER*100 ATTRV
C  mdHandles array of ECS metadata groups in MCF
        CHARACTER*(49)  MDHANDLES(20)
C  names of lobal attributes to store ECS metadata in
        CHARACTER*(MAX_ECS_NAME_L-1) HDFATTNMS(PGSd_MET_NUM_OF_GROUPS)

        CHARACTER*40 attrVal

        INTEGER MODIS_FILE
        INTEGER MCF_FILE

        REAL*8 size

        DATA IDATA/300*1/
        DATA DIMS/15,20,100/
        DATA STA/3*0/
        DATA RANK/3/
        DATA ARRNM /'DATASHORT'/
        DATA GRPNM /' '/
        DATA FILNM /'arrex3.hdf'/
        DATA DIMNM /'RECORD NUMBER'/
        DATA ATTR /MLONG_NAME/
        DATA ATTRV /'This is the attribute value for dimension 2'/
        DATA ATYPE /'CHARACTER*(*)'/
        DATA DTYPE /I32/
        DATA size/100./

        parameter(MODIS_FILE = 201000)
        parameter(MCF_FILE = 10250)

        HDFATTNMS(2) = 'CoreMetadata.0'
        HDFATTNMS(3) = 'ProductMetadata.0'

        NUMHANDLES = 2

        IER = pgs_met_init(MCF_FILE, MDHANDLES)
        if (IER .ne. PGS_S_SUCCESS)
     *   type *, 'error in PGS_MET_Init', IER

        IER = PGS_MET_SetAttr_d(MDHANDLES(2),
     *       'SIZEMBECSDATAGRANULE', size)
        if ( IER .ne. PGS_S_SUCCESS)
     *       type *,'error in PGS_MET_SetAttr', IER

C /*  Try to retrieve this value from the metadata */
        size = 0
C       IER  = PGS_MET_GetSetAttr_d(MDHANDLES(2),
C    *         'SIZEMBECSDATAGRANULE',size)
C       if ( IER .ne. PGS_S_SUCCESS)
C    *       type *,'error in PGS_MET_GetSetAttr', IER
C       type *,'SIZEMBECSDATAGRANULE', size

        IER  = PGS_MET_SetAttr_d(MDHANDLES(2),
     *                      'LONGNAME', attrVal)
        if ( IER .ne. PGS_S_SUCCESS)
     *       type *,'error in PGS_MET_SetAttr', IER

C     Open file
      IER = OPMFIL(FILNM, CREATE_FILE, MODFIL)

        IF(IER.EQ.MAPIOK) THEN
           PRINT *,'Openning of Modis file was successful!'
        END IF

C  Create the array
        PRINT *,'Creating a Data array!'
      IER = CRMAR(MODFIL,ARRNM,GRPNM,DTYPE,RANK,DIMS)
      IF(IER .EQ. MAPIOK) THEN
C     Name the last dimension
           PRINT *,'Naming the last dimension!'
           IER = PMDMIN(MODFIL,ARRNM,GRPNM,0,ATTR,ATYPE,100,ATTRV)
        ENDIF
        IF (IER .EQ. MAPIOK) THEN
           PRINT *,'Writing array to MODIS HDF file!'
C     Re-define the last dimension for writing the array
          DIMS(3) = 1
C     Loop on the last dimension to write the array.
          DO I=1,100
C     and write to the array
             STA(3) = I - 1
             IER = PMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,IDATA)
          END DO
       ENDIF
C     Close HDF file
C       ret_val = cpmfil(metafile, mdhandles, hdfattnms, numhandles)
C       print*,'ret_val = ',ret_val

       IER = CPMFIL(MODFIL, MDHANDLES, HDFATTNMS, NUMHANDLES)
C      IER = CLMFIL(MODFIL)
       IF(IER .EQ. MAPIOK) THEN
          PRINT *,'MODIS file was successfully closed!'
       END IF

C     End of example
       STOP
       END
```

6.5 Example 4: Create an Integer Array in C

This program performs the same operation as the FORTRAN program except that it is written in C. The program demonstrates how to create a 32-bit integer array. The last dimension of the array is named. An array is created using createMODISfile. The last array dimension is named using putMODISdiminfo. The integer array is initialized to one, and then written to the hdf file with putMODISarray. Once the file has been written the file is closed with a call to completeMODISfile.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access r, w, a).
`createMODISarray`	Initializes a MODIS HDF array.
`putMODISarray`	Writes an array or subarray to a MODIS file.
`completeMODISfile`	Completes a new MODIS file.

6.5.1 Source Code Listing for Example 4

```
#include <stdio.h>
#include <string.h>
#include "mapi.h"

/*
**   This example program demonstrates opening a new MODIS HDF file,
**   creating a new data array, writing that array to the HDF file,
**   and closing the HDF file.
*/

main(){
  MODFILE      *modfile;            /* Modis file pointer */
  long dims[3] = {100,20,15};       /* Array dimensions */
  long dnum = 2;                        /* Dimension number to receive info */
  long sta[3] = {0, 0, 0};                /* Array start indices (0-based) */
  long rank = 3;  /* Array rank */
  char dtype[] = I32;   /* Array type, set to M-API macro */
  char arrnm[] = "DATASHORT";       /* Array name */
  char attr[]  = MLONG_NAME;          /* Array attribute, set to M-API macro*/
  char attrv[] = "some really long name";         /* Array attribute value */
  char atype[] = TXT;                  /* Array attribute data type */
  char grpnm[] = "\0";                 /* Group name */
  char fname[] = "arrex4.hdf";         /* File name */
  char acc_mode[] = "w";               /* File access mode */
  long ier;                            /* Error code */
  int i,j,k;                           /* counters */
  int  idata[20][15];               /* Data Array */
  PGSt_MET_all_handles     mdHandles;
  ECSattr_names_for_all_handles HDFattrnms;
  long NumHandles = 0;

  printf(" *** Example4 ***\n");

  memset((void *)idata,1,300);

  /* Create the MODIS-HDF file */
  modfile= openMODISfile(fname,acc_mode);
  if (modfile==NULL) {
    printf("Error opening: %s\n",fname);
    exit(1);
  }else{
    printf("File: %s, opened.\n",fname);
  }
  /* Create array */
  ier = createMODISarray(modfile,arrnm,grpnm,dtype,rank,dims);
  if (ier == MAPIOK)    {
    printf("Array created: %s\n",arrnm);
    /*  Label the last dimension                         */
    ier = putMODISdiminfo(modfile, arrnm, grpnm, dnum,
        attr, atype, strlen(attrv),(void *) attrv);
    if (ier == MFAIL){
      fprintf (stderr, "Error putting diminfo: exiting\n");
      exit(-1);
    }else{
      printf("Dimension %d, putting attribute info: %s =
%s\n",dnum,attr,attrv);
    }
    /*  Re-define the last dimension for writing the array */
    dims[0] = 1;

    /*  Loop on the last dimension to write the array. */
    for (i=0; (i < 100) && (ier == MAPIOK); i++) {

      /*  Set the start index for the last dimension and write to the array */
      sta[0] = i;
      ier = putMODISarray(modfile,arrnm,grpnm,sta,dims,(void *)idata);
    }
    if (ier == MFAIL){
      fprintf (stderr, "Error writing array, exiting\n");
      exit(-1);
    }else{
      printf("Array written: %s\n",arrnm);
    }
  }
  /* Close the MODIS-HDF file */
  ier = completeMODISfile(&modfile, mdHandles, HDFattrnms, NumHandles);
  if (ier == MFAIL){
    fprintf (stderr, "Error closing file, exiting\n");
    exit(-1);
  }else{
    printf("File closed successfully.\n");
  }
  exit(0);
}
/* End of example */
```

6.6 Example 5: Read an Integer Array

This FORTRAN program demonstrates how to read a 32-bit integer array. The HDF file, arrex3.hdf, (created by example3) is openned for reading using OPMFIL. The array data is read in to memory using GMAR. Once the array has been read into memory, the file is closed with a call to CLMFIL.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access r, w, a).
GMAR	Retrieves an array from an HDF file.
CLMFIL	Closes preexisting MODIS file.

6.6.1 Source Code Listing for Example 5

```
      program example5
c EXAMPLE 5:  Read the array from the previous example by
c looping on the second array index, using FORTRAN.

      INCLUDE 'mapi.inc'
c  DATA ARRAY
      INTEGER JDATA(15,100)

C Array Checksum
      INTEGER cksum

c  MODIS FILE POINTER ARRAY
      INTEGER MODFIL(MODFILLEN)

C  DIMENSION ARRAY
      INTEGER DIMS(3)
C  Start indices (0-based) for reading array
      INTEGER STA(3)

C  Error code
      INTEGER IER

C Array and group names
      CHARACTER*20 ARRNM, GRPNM, FILNM

      DATA DIMS/15,20,100/
      DATA STA/3*0/,cksum/0/
      DATA ARRNM/'DATASHORT'/, GRPNM/’  ’  /
      DATA FILNM/'arrex3.hdf'/

      print*,'*** Example5 ***'
```

```
C       Open file
      IER = OPMFIL(FILNM, 'r', MODFIL)
      IF (IER .EQ. MAPIOK) THEN
C       Re-define the second dimension for reading the array
         DIMS(2) = 1
         i=1
C       Loop on the second dimension to read the array.
         DO while (i .le. 20 .and. ier .eq. MAPIOK)
C       Set the start index for the second dimension and read
C       the array
            STA(2) = I - 1
            IER = GMAR(MODFIL,ARRNM,GRPNM,STA,DIMS,JDATA)
            i = i+1
         END DO
         IF (IER .EQ. MAPIOK) THEN
            do m=1,100
       do l=1,15
          cksum = cksum + jdata(l,m)
       end do
            end do
            print*,'Array retrieved: ',arrnm
            print*,'Checksum: ',cksum
         else
            print*,'GMAR: failed @ I=',i
         ENDIF
      ENDIF

C     Close file
      IER = CLMFIL(MODFIL)
      if (ier .ne. MFAIL)then
         print*,'File closed.'
      else
         print*,'Error closing file.'
      endif
C       End of example
      STOP
      END
```

Table 6-1 structure is created with the name "Bolide Heights". The data group argument is set to NULL so the data structure is not placed in any data group 'subdirectory '.

Table 6-1 Sample Data Table
Bolide Heights

Record Number	**Latitude (degrees)**	**Longitude (degrees)**	**Altitude (m)**
Number Type	float32	float32	int32
0	40.2	-77.8	23500
1	-22.8	132.5	37000
2	63.2	93.6	2200

The following example routines create, read, and write MODIS HDF tables.

6.7 Example 6: Create a MODIS HDF Table

This FORTRAN program demonstrates how to create a MODIS HDF table. The table consists of three columns and three rows (see Table 6-1 Sample Data Table). Two columns are real data and one column is integer data. As in the previous examples an HDF file ('tblex6.hdf') is openned for writing using OPMFIL. An HDF table is created using CRMTBL. The table is then written to the HDF file using PMTBL. Once the file has been written the file is closed with a call to CLMFIL.

List of routines called:

Name	**Description**
`OPMFIL`	Opens a MODIS file (file access r, w, a).
`CRMTBL`	Creates a table for accessing.
`PMTBL`	Writes a table to an HDF file.
`CPMFIL`	Completes a new MODIS file.

6.7.1 Source Code Listing for Example 6

```
C     This program will create a modis HDF table called "Bolide Heights"
C     by using CMTBL, then put the 3 records of information in to the
C     table by using PMTBL.
C==================================================================
      program example6
      IMPLICIT NONE
      INCLUDE 'mapi.inc'

C DATA BUFFER
      byte             data1(12)
C MODIS file pointer array
      integer          mfile(MODFILLEN)
C Number of records to access and location of first record to access
      integer            recno, start
C Error code
      integer            ier
C File, table name, table class, and group names
      character*80     filen,tbname,group,classname
C Table field names
      character*80     field
C Data type, using M-API parameter to size string
      character*(3*DATATYPELENMAX) dtype
C Data arrays and type-matched buffers
      real             lat(3), lon(3), f1, f2
      integer          height(3), i3
      integer          i
C mdHandles array of ECS metadata groups in MCF
        character*20 mdhandles

C Names of Global atributes to store ECS metadata in
        character*20 hdfattnms

C  Number of handles
        integer numhandles

data             filen /'tblex6.hdf'/
      data             tbname /'Bolide Heights'/
      data             group /’  ’  /
      data             classname /'Fake Data class'/
      data             lat  /40.50, -22.81, 08.10/
      data             lon  /-80.22, -43.25, 98.32/
      data             height /400, 0, 0/
      data             numhandles/0/

C Map data buffer to data type-matched buffers
      EQUIVALENCE (data1(1), f1)
      EQUIVALENCE (data1(5), f2)
      EQUIVALENCE (data1(9),i3)
```

```
C     Set field names and corresponding data type
      field ='Latitude(degrees),Longitude(degrees),Altitude(m)'
      dtype = R32 //','// R32 //','// I32

      PRINT*,'*** Example6 ***'

C     Open file, using M-API parameter to define file access
      ier = OPMFIL(filen, CREATE_FILE, mfile)

      IF(IER.EQ.MAPIOK) THEN
         PRINT *,'Opened a Modis HDF file!'
      END IF

      if(ier.eq.MAPIOK) then
C       create an HDF table
        ier = CRMTBL(mfile,tbname,classname,group,field,dtype)
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Successfully created a HDF table!'
        END IF

C       Put the data into the modis HDF table. Write 1 record
C       at a time, always append it at the end of the table(-1).
      recno = 1
      start = -1
        do 1 i =  1, 3
           f1 = lat(i)
           f2 = lon(i)
           i3 = height(i)
           if(ier.eq.MAPIOK) then
           ier = PMTBL(mfile,tbname,group,start,recno,data1)
         end if
    1   continue
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'Successfully wrote the table to MODIS HDF file!'
        END IF

c       complete the hdf file
        ier = CPMFIL(mfile, mdhandles, hdfattnms, numhandles)
        IF(IER .EQ. MAPIOK) THEN
           PRINT *,'MODIS HDF file was closed!'
        END IF

      end if
      stop
      end
```

6.8 Example 7: Read HDF Tables in FORTRAN

This FORTRAN program demonstrates how to read an HDF table. The HDF file, tblex6.hdf, (created by example6) is openned for reading using OPMFIL. First, the information about the table is retrieved using GNFLDS. The actual table data is then retrieved using GMTBL. Once the table has been read into memory, the file is closed with a call to CLMFIL.

List of routines called:

Name	Description
OPMFIL	Opens a MODIS file (file access: r, w, a).
GMFLDS	Retrieves info on an HDF table.
GMTBL	Reads an HDF table into memory.
CLMFIL	Closes preexisitng MODIS file.

6.8.1 Source Code Listing for Example 7

```
C    the test program will first open the modis HDF table "Bolide Heights"
C    created by example6.f,  then call GMFLDS and GMTBL to get the
C    table's structural information and then the contents.
      program example7
      IMPLICIT NONE
      INCLUDE 'mapi.inc'

C MODIS file pointer array
      integer         mfile(MODFILLEN)
C Table name, data group name, Table's field names, field data types, and
class
      character*80     tbname, group, fldnm, dtype, classname
C maximum length of character strings returned by GMFLDS
      integer         strln
C Number of records in table, of fields (columns), of first record to read
      integer           recno, fldno, start
C Return code, type-matched buffer, size of read-in buffer
      integer           ret, height, bsize
C Type-matched buffers
      real            lat, lon
C Read-in data buffer
      byte            data(12)
      DATA tbname /'Bolide Heights'/
      DATA group  /’  ’  /
C Map data buffer to data type-matched buffers
      EQUIVALENCE (data(1), lat)
      EQUIVALENCE (data(5), lon)
      EQUIVALENCE (data(9), height)
      print*,'*** Example7 ***'
C     first open the HDF file.
      ret = OPMFIL("tblex6.hdf", "r", mfile)

      if (ret.eq.MAPIOK) then
C       get the number of records and fields in the table, the table's class
C       name, and the names of the fields and their respective data types.
        ret = GMFLDS(mfile, tbname, group, strln, recno,
     *               fldno, fldnm, dtype, classname)
      if (ret.eq.MAPIOK) then
           write(*,*) 'Field Names: ', fldnm
           write(*,*) 'Data Types: ', dtype
         write(*,*) 'Records:'
      end if

C       print the table contents, one record at a time
        do  start = 0, recno-1
           if (ret.eq.MAPIOK) then
             bsize = 12
             ret = GMTBL(mfile,tbname,group,fldnm,start,1,bsize,data)
             if (ret.eq.MAPIOK) write(*,*)  lat, lon, height
          else
             print*,'Error getting table row: ',start
          end if
       end do

C       close the HDF file.
        ret = CLMFIL(mfile)
        IF(RET.NE.MAPIOK)THEN
           PRINT*,'Error closing file.'
        else
           print*,'File closed.'
        endif
      end if
      STOP
      END
```

6.9 Example 8: Read HDF Tables in C

This program performs the same operation as the FORTRAN program except that it is written in C. At the command line the HDF filename, Vdata tablename, and fieldname must be entered. For example: if the file created by example6 is to be read, the following would be entered at the command line "example8 tablex6.hdf Bolide Heights lat". In this example the Bolide Heights table would be accessed and the lat column data would be read into memory. Once all the data are read into memory then the file is closed with a call to closeMODISfile.

List of routines called:

Name	Description
openMODISfile	Opens a MODIS file (file access r, w, a).
getMODISfields	Retrieves HDF table info.
MODISsizeof	Determines size in bytes of an array type.
getMODIStable	Retrieves the HDF table data.
closeMODISfile	Closes a preexisiting MODIS file.

6.9.1 Source Code Listing for Example 8

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mapi.h"

/*
**  This example program demonstrates how to open a MODIS HDF file,
**  read a single field from a given data table from the file,
**  and close the file.
*/

main() {
  MODFILE   *modfile;   /* Modis file pointer */
  void      *data = NULL;     /* Data Array */
  void *dp; /* Data Array scanner */
  char *field_dtype;            /* Table field's data type */
  int  fieldnumber;     /* Table field's number */
  char fldname[] = "\0";              /* Feild name to get */
  char *fieldnames ="\0";           /* Table's field names */
  char *datatypes = "\0";     /* their data types */
  char fname[]="tblex6.hdf";           /* Input file name */
  char vname[]="Bolide Heights";                    /* Vdata name */
  long int stringlen;   /* The length these strings need to be*/
  long int records;     /* Number of records in the table and */
  long int fields;      /* Number of fields (columns) in it */
  char *single_fieldname;     /* Single field name extracted from */
  /* fieldnames       */
  char grpnm[] = "\0";  /* Group name */
  long int size_of_buffer;            /* Size of buffer used to get data */
  long int sta = 0;     /* First record in table to read*/
  long ier = MFAIL;                    /* Error code */
  int i;    /* counter */

  printf(" *** Example8 ***\n");
  /* Open the MODIS-HDF file */
  modfile = openMODISfile(fname, "r");
  if (modfile==NULL){
    printf ("File not found\n");
    exit(1);
  }else{
    printf("File: %s opened.\n",fname);
  }
  /* Get size of strings required to hold field names and data type info */
  /* getMODISfields will return MFAIL because the strings are too short (0) */
  /* but stringlen should return the length required.   */
  /* Note that fieldnames and datatypes must NOT be set to NULL for the  */
  /* string length information to be returned     */

  stringlen = 0;
  (void)getMODISfields(modfile,vname,grpnm,&stringlen,NULL,NULL,
             fieldnames,datatypes,NULL);
  if (stringlen == 0){
    printf ("Table not found\n");
  }else{
    fieldnames = (char *)malloc(stringlen * sizeof(char));
    datatypes = (char *)malloc(stringlen * sizeof(char));

    /* Get dimensional information about the table */
    if (getMODISfields(modfile,vname,grpnm,&stringlen,&records,
             &fields,fieldnames,datatypes,NULL) == MAPIOK){
      printf("Each of the %d records contans these fields: %s\n",
           records, fieldnames);

      /* get the data type of the specified field
       determine the field number of the specified field */
      fieldnumber = 0;
      single_fieldname = strtok(fieldnames,",");
      /* get the data type of the field from the datatypes string */
      field_dtype = strtok(datatypes,",");
      while(single_fieldname != NULL) {
      printf("FieldName: %s \tField dataype:
            %s\n",single_fieldname,field_dtype);

      /* allocate an array to retrieve the data. Note that since the
         data to be retrieved are all of the same data type,
         extracting the data from a generic byte buffer is not required.*/
      size_of_buffer = records * MODISsizeof(field_dtype);
      data = (void *)malloc(size_of_buffer);
```

```
      /* Read field from every record in the table into the data buffer */
      ier = getMODIStable(modfile,vname,grpnm,single_fieldname,
          sta,records,&size_of_buffer,data);
      if(ier == MAPIOK){
        /* List contents of array */
        dp = data;
        for (i= 0; i < records; i++){
          if (strcmp(field_dtype,I8) == 0)
            printf ("record %d = %d\n",i,(int) *((char *)dp));
          else if (strcmp(field_dtype,UI8) == 0)
            printf ("record %d = %u\n",i,(unsigned int) *((unsigned char
                    *)dp));
          else if (strcmp(field_dtype,I16) == 0)
            printf ("record %d = %hd\n",i, *((short int *)dp));
          else if (strcmp(field_dtype,UI16) == 0)
            printf ("record %d = %hu\n",i, *((unsigned short int *)dp));
          else if (strcmp(field_dtype,I32) == 0)
            printf ("record %d = %d\n",i, *((int *)dp));
          else if (strcmp(field_dtype,UI32) == 0)
            printf ("record %d = %u\n",i, *((unsigned int *)dp));
          else if (strcmp(field_dtype,R32) == 0)
            printf ("record %d = %g\n",i, *((float *)dp));
          else if (strcmp(field_dtype,R64) == 0)
            printf ("record %d = %lg\n",i, *((double *)dp));
          else if (strcmp(field_dtype,TXT) == 0)
            printf ("record %d = %c\n",i,(int) *((char *)dp));
          dp = (char *)dp + MODISsizeof(field_dtype);
        }
        free(data);
        if (*fieldnames != '\0'){
          free(fieldnames);
          free(datatypes);
        }
      }else{
        printf("Error calling getMODIStable.\n");
      }
      single_fieldname = strtok(NULL,",");
      field_dtype = strtok(NULL,",");
      }
    }
  }
  /* Close the MODIS-HDF file */
  if ( closeMODISfile(&modfile) == MFAIL ){
    printf ("Error closing file\n");
    exit(1);
  }else{
    printf ("File closed\n");
  }
  exit(0);
}
/* End of example */
```

6.10 Example 9: Read Data from ECS Metadata Files

This C program demonstrates how to read data from an ECS metadata file. (Note: there exists metadata incompatablities between PGS Toolkit v 5.0 and 5.1. The input file “metex9.hdf” supplied with the example programs was created using PGS Tool Kit v 5.1.) AN HDF file “metex9.hdf” is opened for reading using openMODISfile. The ECS metadata is retrieved from the HDF file using getMODISECSinfo. Once the metadata has been retrieved, it is then parsed into individual strings using sustrMODISECSinfo. The HDF file is closed using closeMODISfile.

List of routines called:

Name	Description
`openMODISfile`	Opens a MODIS file (file access r, w, a).
`getMODISECSinfo`	Retrieves the ECS metadata.
`substrMODISECSinfo`	Parses the retrieved ECS metadata.
`closeMODISfile`	Closes a MODIS file.

6.10.1 Source Code Listing for Example 9

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "mapi.h"

main() {
   MODFILE  *modfile;                    /* Modis file pointer */
   long int n_elements=20l;                   /* Number of metadata values
       to extract from value. */
   void *value;
   char access_mode[]="r";
   int  ier;                                  /* Error code */
   int  i;
   long int  n_strings=10;
   char *substr[10];
   int  size = 256;

   char filename[]="metex9.hdf";              /* Input file */
   /*  NOTE:  This file included is for use with PGS Toolkit v 5.1
       it is a documented fact that metadata incompatibilities
       exist between PGS Toolkit 5.0 and 5.1, in this case older
       files generated w/ 5.0 may not work if M-API was linked with TK 5.1  */
   char PVLAttrName[]="CoreMetadata.0";       /* PVL Attribute name input */
   char parmName[] = "SHORTNAME";              /* parameter name */
   char data_type[] = "char *";             /* Data type of
             the parameter value */
```

```
printf(" *** Example 9 ***\n");
/* Allocate memory for value. */
value = (void *)malloc(size);

/* Open the MODIS-HDF file */
modfile= openMODISfile(filename, access_mode);

if (modfile==NULL) {
  printf("Unable to open file %s\n",filename);
  exit(1);
}else{
  printf("Opening the file\n");
}
ier = getMODISECSinfo(modfile, PVLAttrName, parmName, data_type,
   &n_elements, value);
if (ier==MFAIL){
  printf("ier(getMODISECSinfo) = %d\n",ier);
  printf("n_elements = %ld\n",n_elements);
  printf("data_type = %s\n",data_type);
  printf("PVLAttrName = %s\n",PVLAttrName);
  printf("parmName = %s\n",parmName);
}else{
  printf("n_elements = %ld\n",n_elements);
  printf("data_type = %s\n",data_type);
}
if ( (ier == MAPIOK) && ( n_elements != 0 ) ){
  if ( strcmp(data_type, I32) == 0 )
    for (i=0; i < n_elements; i++)
      printf("value = %ld ",((int32 *)value)[i]);
  if (strcmp(data_type, R32) == 0)
    for (i=0; i < n_elements; i++)
      printf("value = %f ",((float32 *)value)[i]);
  if (strcmp(data_type, R64) == 0)
    for (i=0; i < n_elements; i++)
      printf("value = %f ",((float64 *)value)[i]);
  printf("\n");
  if (strcmp(data_type, TXT) == 0){
    ier = substrMODISECSinfo(value,n_elements,&n_strings,substr);
    if (ier==MFAIL){
    printf("ier(substrMODISECSinfo) = %d\n",ier);
    printf("Error printing the substrings\n");
    }else{
    printf("n_strings = %d\n",n_strings);
    printf("string(s) = \n");
    for (i=0;i<n_strings;i++)
      printf("%s\n",substr[i]);
    }
  }
}

/* Close the MODIS-HDF file */
ier = closeMODISfile(&modfile);
if (ier == MFAIL){
  printf("Error closing file\n");
```

```
     exit(1);
   }else{
     printf("File closed.\n");
     exit(0);
   }
}
/* End of example */
```

APPENDIX A: ACRONYMS

ABI	Application Binary Interface
ANSI	American National Standards Institute
ASCII	American Standard for Computer Information Interchange
ATBD	Algorithm Theoretical Basis Document
AVHRR	Advanced Very High Resolution Radiometer
DAAC	Distributed Active Archive Center
DEC	Digital Equipment Corporation
DIF	Data Interchange Format
ECS	EOSDIS Core System
EOS	Earth Observing System
EOSDIS	Earth Observing System Data and Information System
ESDIS	Earth Science Data and Information System
FTP	File Transfer Protocol
GCMD	Global Change Master Directory
GSC	General Sciences Corporation
GSFC	Goddard Space Flight Center
HDF	Hierarchical Data Format
IDL	Interactive Data Language
I/O	Input/Output
IP	Internet Protocol
L1	Level 1
L1B	Level 1B
L2	Level 2
L3	Level 3
M-API	MODIS Applications Programming Interface
MCF	Metadata Configuration File
MODIS	Moderate Resolution Imaging Spectroradiometer
NASA	National Aeronautics and Space Administration
NCSA	National Center for Supercomputing Applications
ODL	Object Description Language

PCF	Process Control Files
PGE	Product Generation Executive
PVL	Parameter Value Language
QA	Quality Assurance
SAIC	Science Applications International Corporation
SCF	Science Computing Facilities
SD	Scientific Data
SDP	Science Data Processing
SDPS	Science Data Processing Segment
SDS	Scientific Data Set
SDST	Science Data Support Team
SeaWiFS	Sea-viewing Wide Field-of-view Sensor
SGI	Silicon Graphics, Inc.
SSTG	Science Software Transfer Group
STM	Science Team Member
TLCF	Team Leader Computing Facility
TRMM	Tropical Rainfall Measuring Mission
URLs	Uniform Resoure Locators
V	Vgroup
VS	Vdata Set
WWW	World Wide Web

APPENDIX B: M-API-SUPPLIED CONSTANTS AND MACROS

The following tables show the constants that are found in the mapi.h (C) and mapi.inc (FORTRAN):

Table B-1 Data Type Constants

Metadata Name/Description	M-API Constant
array structure and dimension label string	`MLONG_NAME`
array structure and dimension units string	`MUNITS`
array structure and dimension format string	`MFORMAT`
array structure coordinate system string	`MCOORD_SYS`
array structure Calibration factor	`MSLOPE`
array structure Calibration factor error	`MSLOPE_ERROR`
array structure uncalibrated offset	`MOFFSET`
array structure uncalibrated offset error	`MOFFSET_ERROR`
array structure uncalibrated data HDF number type	`MNUM_TYPE`
standard data valid range (Sdgetrange)[minimum,]	`MDATA_RANGE`
array structure Fill Value	`MFILL_VALUE`
ECS inventory metadata global attribute name	`MECS_CORE`
ECS archive metadata global attribute name	`MECS_ARCHIVE`
'Same as above' - returned for Backward compatibility	`MECS_PRODUCT`

Table B-2 ECS Global Inventory Metadata Names

Note: User should refer to a particular file specification for a more precise layout of the metadata for a product.

Metadata Name/Description	M-API Constant
HDFattrNames = **MECS_CORE**	
References to all ancillary input files, i.e. all input files other than MODIS products.	MCORE_ANCIL_POINTER
Indicates the results of QA performed during product generation.	MCORE_AUTO_QUALITY
Easternmost longitude of the granule spatial coverage.	MCORE_EAST_BOUND
Flag indicating whether points are on an inner (exclusion) G-ring.	MCORE_EXCLUS_GRING_FLG
Self-reference to granule. For V1, this field should be identical to MODISPRODUCTFILENAME.	MCORE_GRAN_POINTER
Latitudes of a series of points representing the perimeter of the granule spatial coverage (i.e., corners).	MCORE_GRING_POINT_LAT
Longitudes of a series of points representing the perimeter of the granule spatial coverage.	MCORE_GRING_POINT_LON
Sequence numbers corresponding to perimeter latitudes and longitudes.	MCORE_GRING_POINT_NUM
References to other MODIS product granules used as input for this product.	MCORE_INPUT_POINTER
A descriptive name for the data collection.	MCORE_LONG_NAME
Northernmost latitude of the granule spatial coverage.	MCORE_NORTH_BOUND
The granule level flag applying both generally to the granule and specifically to the parameters at the granule level. When applied to a parameter, the flag refers to the quality of that parameter in the granule.	MCORE_OPER_QUAL_FLAG
Number of satellite orbit during which the granule data were collected.	MCORE_ORBIT_NUM
Reference to processing history file.	MCORE_HISTORY_POINTER
Value indicating the percent of interpolated data in the granule	MCORE_PERCENT_INTERP
Value indicating the percent of missing data in the granule.	MCORE_PERCENT_MISSING
Value indicating the percent of data in the granule outside of acceptable limits.	MCORE_PERCENT_OUT
A text explanation of the criteria used to set each quality lag; including thresholds or other criteria.	MCORE_QUAL_EXPL
The date and time when the temporal coverage period of this granule began.	MCORE_RANGE_START
The date and time when the temporal coverage period of this granule ended.	MCORE_RANGE_END
Indicator of what reprocessing is planned for the granule.	MCORE_TO_BE_REDONE
Indicator of the reprocessing status of the granule.	MCORE_ACTUALLY_REDONE

Metadata Name/Description	M-API Constant
The granule level flag applying to the granule and to the parameters at the granule level. When applied to a parameter, the flag refers to the quality of that parameter in the granule.	MCORE_SCIENCE_QUAL_FLG
The identifier for the data collection.	MCORE_SHORT_NAME
The size of the data granule in megabytes.	MCORE_SIZE_OF_GRANULE
Southernmost latitude of the granule spatial coverage.	MCORE_SOUTH_BOUND
Westernmost longitude of the granule spatial coverage.	MCORE_WEST_BOUND
The MODIS filename for this granule.	MPROD_FILENAME
MODIS mode of operation.	MPROD_OPERATIONMODE
This field contains the date and time the process that created this file was started.	MPROD_PROC_DATE_TIME
The SPSO parameters for all data contained in this file, as listed in the SPSO database.	MPROD_SPSO_PARAM
The number of this MODIS granule.	MPROD_GRANULE_NUM
HDFattrNames = **MECS_PRODUCT**	
The date this algorithm package version successfully passed AI&T procedures and was accepted as an ECS standard algorithm.	MPROD_ALGO_PCK_ACPT_DATE
This specifies the maturity of the algorithm package	MPROD_ALGO_PACK_MAT_CODE
Algorithm package name	MPROD_ALGO_PACK_NAME
The version of the algorithm package.	MPROD_ALGO_PACK_VER
The long name by which the instrument is known.	MPROD_INSTR_NAME
The short name assigned to the platform carrying the instrument.	MPROD_PLATFORM_SHORT_NAM
DAAC where product is processed.	MPROD_PROC_CENTER

Table B-3 Level 1A Macros

Metadata Name/Description	M-API Constatnt
MOD01_L1A	MOD01_L1A
Scan number	M01SCAN_NUMBER
Frame count array	M01FRAME_COUNT_ARRAY
Scan Type	M01SCAN_TYPE
SD start time	M01SD_START_TIME
SRCA start time	M01SRCA_START_TIME
BB start time	M01BB_START_TIME
SV start time	M01SV_START_TIME
EV start time	M01EV_START_TIME
SRCA calibration mode	M01SRCA_CALIBRATION_MODE
Packet scan count	M01PACKET_SCAN_COUNT
CCSDS Application Identifier	M01CCSDS_APID
Packet Quick Look flag	M01PACKET_QL
Mirror side	M01MIRROR_SIDE
Scan quality array	M01SCAN_QUALITY_ARRAY
Earth sector Pixel quality	M01EV_PIX_QUAL
SD sector Pixel quality	M01SD_PIX_QUAL
SRCA sector Pixel quality	M01SRCA_PIX_QUAL
BB sector Pixel quality	M01BB_PIX_QUAL
SV sector Pixel quality	M01SV_PIX_QUAL
Bands 1 and 2	M01EV_250M
Bands 3 through 7	M01EV_500M
Bands 8 through 19	M01EV_1KM_DAY
Bands 20 through 36	M01EV_1KM_NITE
Bands 1 and 2	M01SD_250M
Bands 3 through 7	M01SD_500M
Bands 8 through 19	M01SD_1KM_DAY
Bands 20 through 36	M01SD_1KM_NITE
Bands 1 and 2	M01SRCA_250M
Bands 3 through 7	M01SRCA_500M
Bands 8 through 19	M01SRCA_1KM_DAY
Bands 20 through 36	M01SRCA_1KM_NITE
Bands 1 and 2	M01BB_250M
Bands 3 through 7	M01BB_500M
Bands 8 through 19	M01BB_1KM_DAY
Bands 20 through 36	M01BB_1KM_NITE
Bands 1 and 2	M01SV_250M
Bands 3 through 7	M01SV_500M
Bands 8 through 19	M01SV_1KM_DAY
Bands 20 through 36	M01SV_1KM_NITE

Metadata Name/Description	M-API Constatnt
Eng. packet 1 data	M01RAW_ENG_PKT_1
Eng. packet 2 data	M01RAW_ENG_PKT_2
Mem. packet 1 data	M01RAW_MEM_PKT_1
Mem. packet 2 data	M01RAW_MEM_PKT_2
FPA DCR offset data	M01FPA_DCR_OFFST
FAM Registration sample Delays	M01FAM_SAMP_DELAY
Raw mirror encoder data	M01RAW_MIR_ENC
Current/Prior HK Telem	M01RAW_HK_TELEM
Sci Eng Data	M01RAW_SCI_ENG
Parameter Table	M01RAW_PARAM
View Sector Start	M01RAW_VS_START
CP Event Log	M01RAW_CP_EVENT
FR Event Log	M01RAW_FR_EVENT
Raw s/c ancill data	M01RAW_SC_ANCIL
Dump Request Info	M01RAW_DUMP_REQ
Dump Data	M01RAW_DUMP_DATA
FPA/AEM Config	M01FPA_AEM_CONFIG
FPA Use	M01FPA_USE

Table B-4 L1B/Geolocation Macros

Metadata Name/Description	M-API Constant
Product type identifier	M02_PROD_ID
Software Version	M02VERSION
Number of Scans	M02NUMBER_OF_SCANS
Number of Day mode scans	M02NUMBER_OF_DAY_SCANS
Number of Night mode scans	M02NUMBER_OF_NIGHT_SCANS
Max Total Frames	M02MAX_TOTAL_FRAMES
Max Earth View Frames	M02MAX_EARTH_FRAMES
Max SD Frames	M02MAX_SD_FRAMES
Max SRCA Frames	M02MAX_SRCA_FRAMES
Max BB Frames	M02MAX_BB_FRAMES
Max SV Frames	M02MAX_SV_FRAME
Scan types in product	M02SCAN_TYPES
Dead MODIS Detectors	M02DEAD_DETECTORS
Noisy MODIS Detectors	M02NOISY_DETECTORS
Dead Thermistors	M02DEAD_THERMISTORS
Noisy Thermistors	M02NOISY_THERMISTORS
250 M Band Numbers for Reflected Solar Bands	M02_250M_BAND_NUMS
500 M Band Numbers for Reflected Solar Bands	M02_500M_BAND_NUMS
1000 M Band Numbers for Reflected Solar Bands"	M02_1000M_REF_BAND_NUMS
Incomplete Scans	M02PARTIAL_SCANS
Missing Packets	M02MISSING_PACKETS
Packets with bad CRC	M02BAD_PACKETS
Discarded Packets	M02DISCARD_PACKETS
Swath Vgroup	M02SWATHWATH
num_scale_factors	M02NUM_SCALE_FACTORS
40*nscans	M02_40NSCANS
20*nscans	M02_20NSCANS
10*nscans	M02_10NSCANS
nscans	M02_NSCANS
40*nRefSBscans	M02_40NREFSBSCANS
20*nRefSBscans	M02_20NREFSBSCANS
10*nRefSBscans	M02_10NREFSBSCANS
Band_250M	M02BAND_250M
Band_500M	M02BAND_500M
Band_1KM_RefSB	M02BAND_1KM_REFSB
Band_1KM_Emissive	M02BAND_1KM_EMIS
4*BB frames	M02_4BB_FRAMES
2*BB frames	M02_2BB_FRAMES
BB frames	M02_BB_FRAMES
4*EV frames	M02_4EV_FRAMES

Metadata Name/Description	M-API Constant
2*EV frames	M02_2EV_FRAMES
EV frames	M02_EV_FRAMES
4*SD frames	M02_4SD_FRAMES
2*SD frames	M02_2SD_FRAMES
SD frames	M02_SD_FRAMES
4*SRCA frames	M02_4SRCA_FRAMES
2*SRCA frames	M02_2SRCA_FRAMES
SRCA frames	M02_SRCA_FRAMES
4*SV frames	M02_4SV_FRAMES
2*SV frames	M02_2SV_FRAMES
SV frames	M02_SV_FRAMES
Instrument Data Stored as Scientific Data Sets	M02SLOPE_AND_OFFSET
Black Body 250M Reflected Solar Bands Scaled Integer Radiance	M02BB_250
Black Body 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02BB_250_UNCERT
Earth View 250M Reflected Solar Bands Scaled Integer Radiance	M02EARTH_RAD_250
Earth View 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02EARTH_RAD_250_UNCERT
Solar Diffuser 250M Reflected Solar Bands Scaled Integer Radiance	M02DIFFUSER_250
Solar Diffuser 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02DIFFUSER_250_UNCERT
RCA 250M Reflected Solar Bands Scaled Integer Radiance	M02SRCA_250
SRCA 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SRCA_250_UNCERT
Space View 250M Reflected Solar Bands Scaled Integer Radiance	M02SPACE_250
Space View 250M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SPACE_250_UNCERT
Black Body 500M Reflected Solar Bands Scaled Integer Radiance	M02BB_500
Black Body 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02BB_500_UNCERT
Earth View 500M Reflected Solar Bands Scaled Integer Radiance	M02EARTH_RAD_500
Earth View 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02EARTH_RAD_500_UNCERT
Solar Diffuser 500M Reflected Solar Bands Scaled Integer Radiance	M02DIFFUSER_500
Solar Diffuser 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02DIFFUSER_500_UNCERT

Metadata Name/Description	M-API Constant
SRCA 500M Reflected Solar Bands Scaled Integer Radiance	M02SRCA_500
SRCA 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SRCA_500_UNCERT
Space View 500M Reflected Solar Bands Scaled Integer Radiance	M02SPACE_500
Space View 500M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SPACE_500_UNCERT
Black Body 1000M Reflected Solar Bands Scaled Integer Radiance	M02BB_1000
Black Body 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02BB_1000_UNCERT
Black Body 1000M Emissive Bands Scaled Integer Radiance	M02BB_EMIS_1000
Black Body 1000M Emissive Bands Scaled Integer Radiance Uncertainty	M02BB_EMIS_1000_UNCERT
Earth View 1000M Reflected Solar Bands Scaled Integer Radiance	M02EARTH_RAD_1000
Earth View 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02EARTH_RAD_1000_UNCERT
Earth View 1000M Emissive Bands Scaled Integer Radiance	M02EARTH_EMIS_RAD_1000
Earth View 1000M Emissive Bands Scaled Integer Radiance Uncertainty	M02EARTH_EMIS_RAD_1000_UNCERT
Solar Diffuser 1000M Reflected Solar Bands Scaled Integer Radiance	M02DIFFUSER_1000
Solar Diffuser 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02DIFFUSER_1000_UNCERT
Solar Diffuser 1000M Emissive Bands Scaled Integer Radiance	M02DIFFUSER_EMIS_1000
Solar Diffuser 1000M Emissive Bands Scaled Integer Radiance Uncertainty	M02DIFFUSER_EMIS_1000_UNCERT
SRCA 1000M Reflected Solar Bands Scaled Integer Radiance	M02SRCA_1000
SRCA 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SRCA_1000_UNCERT
SRCA 1000M Emissive Bands Scaled Integer Radiance	M02SRCA_EMIS_1000
SRCA 1000M Emissive Bands Scaled Integer Radiance Uncertainty	M02SRCA_EMIS_1000_UNCERT
Space View 1000M Reflected Solar Bands Scaled Integer Radiance	M02SPACE_1000
Space View 1000M Reflected Solar Bands Scaled Integer Radiance Uncertainty	M02SPACE_1000_UNCERT
Space View 1000M Emissive Bands Scaled Integer Radiance	M02SPACE_EMIS_1000

Metadata Name/Description	M-API Constant
Space View 1000M Emissive Bands Scaled Integer Radiance Uncertainty	M02SPACE_EMIS_1000_UNCERT
Earth View 250M Reflected Solar Bands Scaled Integer Reflectance	M02EARTH_REFL_250
Earth View 250M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	M02EARTH_REFL_250_UNCERT
Earth View 500M Reflected Solar Bands Scaled Integer Reflectance	M02EARTH_REFL_500
Earth View 500M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	M02EARTH_REFL_500_UNCERT
Earth View 1000M Reflected Solar Bands Scaled Integer Reflectance	M02EARTH_REFL_1000
Earth View 1000M Reflected Solar Bands Scaled Integer Reflectance Uncertainty	M02EARTH_REFL_1000_UNCERT
Eng. Packet 1 Data	M02ENG_PKT_1
Eng. Packet 2 Data	M02ENG_PKT_2
Mem. Packet 1 Data	M02MEM_PKT_1
Mem. Packet 2 Data	M02MEM_PKT_2
FPA DCR offset Data	M02FPA_DCR_OFFSET
FAM Registration Sample Delays	M02FAM_DELAY
Raw Mirror Encder Data	M02MIRROR_ENCODER
Current/Prior HK Telemtry	M02HK_TELEM
Science Engineering Data	M02SCI_ENG
Parameter Table	M02PARM_TABLE
View Sector Start	M02VIEW_START
CP Event Log	M02CP_LOG
FR Event Log	M02FR_LOG
Raw S/C Ancillary Data	M02SC_ANCIL
Dump Request Information	M02DUMP_REQUEST
Dump Data	M02DUMP
Instrument Telemetry	M02INSTR_TELEM
Level 1B Swath Metadata Written as Vdata	M02SWATH_MD
Scan Number /* I32 */	M02SW_SCAN_NO
Total Frames /* I32 */	M02SW_TOT_FRAMES
EV Frames /* I32 */	M02SW_EV_FRAMES
SD Frames /* I32 */	M02SW_SD_FRAMES
SRCA Frames /* I32 */	M02SW_SRCA_FRAMES
BB Frames /* I32 */	M02SW_BB_FRAMES
SV Frames /* I32 */	M02SW_SV_FRAMES
Scan Type /* TXT */	M02SW_SCAN_TYPE
Scan Start Time /* F64 */	M02SW_SCAN_START
Mirror Side /* I32 */	M02SW_MIR_SIDE
Missing Packets /* I32 */	M02SW_MISS_PKTS

Metadata Name/Description	M-API Constant
Packets With Bad CRC /* I32 */	M02SW_BAD_PKTS
Discarded Packets /* I32 */	M02SW_DISC_PKTS
Moon in SV Port /* I32 */	M02SW_MOON_OBS
On-Orbit Manuever /* TXT */	M02SW_MANEUVER
No. SV Outliers /* I32 */	M02SW_NUM_SV_OUTLIERS
No. BB Outliers /* I32 */	M02SW_NUM_BB_OUTLIERS
No. thermistor outliers /* I32 */	M02SW_NUM_THERM_OUTLIERS
Product type identifier	M03_PROD_ID
Mirror wedge angle bias (V1)	M03V1
Mirror axis error bias (gamma)	M03GAMMA
Nominal mirror rotation rate	M03MIR_RATE
Sample interval for 1 km bands	M03T_FRAME
Mirror side 1 encoder-to-angle conversion coefficients (quadratic)	M03POLY_M1
Mirror side 2 encoder-to-angle conversion coefficients (quadratic)	M03POLY_M2
Spacecraft-to-instrument transformation matrix	M03T_INST2SC
Instrument-to-mirror transformation matrix	M03T_MIRR2INST
Instrument-to-telescope transformation matrix	M03T_TEL2INST
Focal length for detectors (0 is ideal)	M03FOCAL_LENGTH
Y offsets for ideal detectors	M03Y_OFFSET
X offsets for 1 km bands	M03X_OFF1KM
Y offsets for 1 km bands	M03Y_OFF1KM
X offsets for 500 m bands	M03X_OFF500
Y offsets for 500 m bands	M03Y_OFF500
X offsets for 250 m bands	M03X_OFF250
Y offsets for 250 m bands	M03Y_OFF250
Band readout times relative to ideal band	M03T_OFFSET
Scan number in granule	M03S_NUM
Number of frames in scan	M03NFRAMES
Scan start time (TAI)	M03SSTIME
Scan center time (TAI)	M03SCTIME
Mirror side	M03MSIDE
Scan quality flags (TBD)	M03SFLAGS
ECR orbit position at scan center time	M03ORB_POS
ECR orbit velocity at scan center time	M03ORB_VEL
ECR-to-instrument frame transformation matrix at scan center time	M03T_INST2ECR
Spacecraft angular velocity in instrument frame	M03ANG_VEL
Unit Sun vector in ECR frame at scan center time	M03SUN_REF
Number of mirror encoder samples for this scan	M03NUM_IMPULSE

Metadata Name/Description	M-API Constant
Mirror angles from encoder data	M03IMPULSE_ENC
Mirror encoder sample times from start of scan	M03IMPULSE_TIME
Band-to-band geometric correction coefficients (based upon algorithm in ATBD)	M03BAND_GEO
Geodetic longitude	M03LONGITUDE
Geodetic latitude	M03LATITUDE
Height above ellipsoid	M03HEIGHT
Sensor zenith	M03SENSOR_ZEN
Sensor azimuth	M03SENSOR_AZ
Range (pixel to sensor)	M03RANGE
Solar zenith	M03SOLAR_ZENITH
Solar azimuth	M03SOLAR_AZIMUTH
Geolocation flags	M03GFLAGS

Table B-5 Atmosphere Macros

Metadata Name/Description	Constant
MOD04_L2	M04L2_PROD_ID
MOD05_L2	M05L2_PROD_ID
MOD06_L2	M06L2_PROD_ID
MOD07_L2	M07L2_PROD_ID
MOD08_L2	M08L2_PROD_ID
MOD30_L2	M30L2_PROD_ID
MOD35_L2	M35L2_PROD_ID
MOD38_L2	M38L2_PROD_ID
1-km_Pixels_Per_Scan_Line	MAPIXELS_PER_SCAN
1-km_Scan_Lines_Per_Granule	MALINES_PER_GRANULE
GMT Time of observation in milliseconds	MAGMT
Corner latitude of 10x10 pixel array	MACORNER_LAT
Corner longitude of 10x10 pixel array	MACORNER_LON
Scanline number through center of 5x5 pixel array	MASCANLINE_NO
Frame number of center pixel in 5x5 array	MAPIXEL_NO
Satellite zenith angle at midpoint of 5x5 array	MAZENITH_SAT
Solar zenith angle at midpoint of 5x5 array	MAZENITH_SOLAR
Index indicating the surface geography type as either Water(0) or Land(1)	MAGEO_FLAG
Surface temperature at midpoint of 5x5 pixel array	MATEMP_SFC
Surface pressure at midpoint of 5x5 pixel array	MAPRES_SFC
Estimated tropopause height	MATROPOPAUSE
long_name	MALONG_NAME
sampling_factor"	MASAMPLING
scale_factor	MASCALE
add_offset	MAOFFSET
units	MAUNIT
valid_range	MARANGE
Number Of Cells Across Swath	MACELLS_ACROSS
Number Of Cells Along Swath	MACELLS_ALONG
Pixels Per Scan Line	MAPIXELS
Number of Scan Lines	MASCANLINE
Number of Bands	M04BANDS
Observed land reflectances averaged on 10x10 1-km pixel array	M04LAND_REFLS
Land aerosol optical thickness (AOT) for continental model	M04LAND_OPT_THICK
Standard deviation of observed land reflectances	M04LAND_REFLS_DEV
Land AOT for corrected model	M04LAND_OPT_THICK_COR

Metadata Name/Description	Constant
Aerosol path radiance ratio (continental model) of red to blue channel (band 3/band 1)	M04LAND_RADIANCE_RATIO
Relative contribution of smoke/sulfate particles to dust in the computation of the aerosol optical depth	M04LAND_CONTRIBUTION
Number of Clear Land Pixels in Band 3	M04LAND_PIXELS_B3
Number of Clear Land Pixels in Band 1	M04LAND_PIXELS_B1
Identification of retrieval procedure	M04LAND_PROC_ID
Aerosol type in one of four categories: continental, dust, sulfate, and smoke	M04LAND_AERO_TYPE
Aerosol land error flag	M04LAND_ERROR
Ocean AOT at 0.55 micron on 10x10 1-km pixel array	M04OCEAN_OPT_THICK
Small-particle ocean AOT at 0.55 micron on 10x10 pixel array	M04OCEAN_OPT_THICK_S
Large-particle ocean AOT at 0.55 micron on 10x10 pixel array	M04OCEAN_OPT_THICK_L
Weight factor for combining large and small aerosol modes during retrieval. This parameter minimizes the least-squares error summed over spectral bands	M04OCEAN_ERROR
Solution number from 1 to 36	M04OCEAN_SOLUTION
Observed ocean reflectances averaged on 10x10 1-km pixel array	M04OCEAN_REFLS
Look-Up Table of Aerosol Model Parameters and Values Vdata	M04AEROSOL_LUT
small mode aerosol mean radius	M04LUT_RGSS
large mode aerosol mean radius	M04LUT_RGSB
standard deviation of small mode radius	M04LUT_SIGMAS
standard deviation of large mode radius	M04LUT_SIGMAB
CCN	M04LUT_CCNS
small mode extinction coefficient for 5 wavelengths	M04LUT_EXTS
large mode extinction coefficient for 5 wavelengths	M04LUT_EXTB
moments order 1-4 of small mode particle radius	M04LUT_MOMENTS
moments order 1-4 of large mode particle radius	M04LUT_MOMENTB
small mode backscatter ratio for 5 wavelengths	M04LUT_BACKSCTS
large mode backscatter ratio for 5 wavelengths	M04LUT_BACKSCTB
small mode asymmetry factor for 5 wavelengths	M04LUT_ASSYMS
large mode asymmetry factor for 5 wavelengths	M04LUT_ASSYMB
small mode albedo for 5 wavelengths	M04LUT_ALBEDOS
large mode albedo for 5 wavelengths	M04LUT_ALBEDOB
Total column water vapor amounts over clear land, and cloud scenes over land and ocean	M05WATER_VAPOR
Index indicating cloud(0), no cloud(1), or cloud/no cloud determination not made(-1)	M05CLOUD_QUAL
Number_Of_1-km_Bands	M06BANDS

Metadata Name/Description	Constant
Number of Channel Indices	M06CHANNEL_IND
Number of Channel Differences	M06CHANNEL_DIFF
Brightness temperatures for IR channels 27 - 36 at 5x5 1-km pixel resolution	M06BRIGHT_TEMP
Sufficient number of cloudy pixels (0) or too few cloudy pixels (1) to be able to process 5x5 pixel array	M06PROCESS_FLAG
Spectral cloud forcing for IR channels 29, and 31 - 36	M06CLOUD_FORCING
value to indicate the method of cloud height determination	M06METHOD
Cloud top effective emissivity	M06EMISSIVITY_CT
Cloud top pressure	M06PRES_CT
Cloud top temperature	M06TEMP_CT
Cloud fraction at 5x5 1-km pixel resolution	M06FRACTION
Separate cloud top pressure estimates from five radiances ratios	M06PRES_CT_RATIO
Cloud top pressure from IR window	M06PRES_CT_IR
Surface type index	M06SFC_TYPE
Radiance variance for channels 29, 31, and 32	M06RADIANCE
Brightness temperature differences between IR channels 29, 31, and 32	M06BRIGHT_TEMP_DIFF
Cloud thermodynamic phase derived from infrared retrieval algorithm	M06PHASE_IR
Effective particle radius at 1-km resolution	M06EFF_RADIUS
Cloud optical thickness at 1-km pixel resolution	M06CLOUD_OPT_THICK
Cloud thermodynamic phase derived from visible/SW infrared retrieval algorithm	M06PHASE_VIS
Statistics at 1-km pixel resolution	M06STATISTICS
Total Colume Ozone at 5x5 1-km pixel resolution	M07TOTAL_OZONE
Total Totals Atmospheric Stability Index	M08TOTALS
Lifted Index Atmospheric Stability Index	M08LIFTED_INDEX
K Index Atmospheric Stability Index	M08K_INDEX
Number Of Channels	M30CHANNELS
Brightness temperatures for IR channels 20, 22-25, and 27-36	M30BRIGHT_TEMP
Guess temperature profile for 20 vertical levels	M30TEMP_PROF
Guess dewpoint temperature profile for 15 vertical levels	M30DEWP_TEMP_PROF
Rretrieved temperature profile for 20 vertical levels	M30RETR_TEMP_PROF
Rretrieved dewpoint temperature profile for 15 vertical levels	M30RETR_DEWP_TEMP_PROF
Index of pressure levels for the 15 vertical levels	M30PRESS_LEVEL
Bit field mask containing the results of visible and infrared radiance cloud/no cloud tests	M35CLOUD_MASK

Metadata Name/Description	Constant
Cell Frame Number	M38CELL_FRAME
Cell Line Number	M38CELL_LINE
Atmospheric Water Vapor Parameter at 5x5 1-km pixel resolution	M38WATER_VAPOR

Table B-6 Ocean Macros

Metadata Name/Description	M-API Constant
MOD27 HDF output file	M27_PROD_ID
output_file_name	M27O_F_NAME
output_file_logical file number	M27O_F_L_F_NUM
units_of_output_file_logical_file_number	U_O_O_F_L_F_NUM
product_name	M27P_NAME
statistics_file_name	M27S_F_NAME
product_sum_total_over_all_regions	M27P_SUM
units_of_product_sum_total_over_all_regions	M27U_O_P_SUM
product_variance_total_over_all_regions	M27P_VAR
units_of_product_variance_total_over_all_regions	M27P_O_P_VAR
product_area_total_over_all_regions	M27P_AREA
units_of_product_area_total_over_all_regions	M27U_O_P_AREA
square km	M27SQKM
number_of_regions_for_product	M27P_NREGS
coordinate_system	M27COORD_SYS
units_of_coordinate_system	M27U_O_COORD_SYS
range_of_coordinate_system	M27R_O_COORD_SYS
character_counter	M27KCHAR
region_counter	M27JREG
limit_of_region_counter	M27KLIM
function_order_counter	M27KORD
product_cell_counter	M27KCELLS
name_of_regions	M27NAME_R
limit_of_regions-deg_lat_and_deg_long	M27LIM_R
area_of_regions-km_squared	M27AREA_R
independent_variables_of_regions	M27IV_R
functions_used_in_regions	M27FUNCTIONS_R
order_of_functions_used_in_regions	M27ORD_R
coefficients_used_in_regions	M27COEFF_R
error_in_regions-gr_per_m3_per year	M27ERR_R
sum_in_regions-gr_per_m3_per_year	M27SUM_R
variance_in_regions-gr2_per_m6_per_year2	M27VAR_R
product_y-gr_per_m3_per_year	M27P_Y
product_error_ey-gr_per_m3_per_year	M27P_EY

Table B-7 Land Macros

Metadata Name/Description	M-API Constant
Pixels_per_scan_line	MLPIXELS_PER_SCAN
Number_of_scan_lines	MLNUMBER_OF_LINES
Pixels_per_line	MLPIXELS_PER_LINE
Lines_per_tile	MLLINES_PER_TILE
Total_observations	MLTOTAL_OBSERVATIONS
Num_parameters	MLNUMBER_OF_PARAMS
Maximum_observations	MLMAX_OBSERVATIONS
Number_of_granules	MLNUMBER_OF_GRANULES
Granule_IDs	MLGRANULE_IDS
File_Format	MLFILE_FORMAT
Parameter1	MLPARM1
Parameter2	MLPARM2
Parameter3	MLPARM3
Parameter4	MLPARM4
Parameter5	MLPARM5
Parameter6	MLPARM6
Parameter7	MLPARM7
Year	MLYEAR
Day_of_year	MLDOY
nrow	MLNUMBER_OF_ROWS
nest_lev	MLNEST_LEVEL
ref_lon_in_deg	MLREF_LONGITUDE
ang_size_in_arcsec	MLANGULAR_SIZE
irow_start	MLIROW_START
ncol_max	MLNCOL_MAX
itile_horiz	MLITILE_HORIZ
itile_vert	MLITILE_VERT
ntile_horiz	MLNTILE_HORIZ
ntile_vert	MLNTILE_VERT
L2G number of observations per pixel contained within L2G file	MLNUMBER_OF_OBS
The number of columns in the full ISCCP grid for each row (line) contained within the L2G file	MLNUMBER_OF_COLS
The start column in the full ISCCP grid for each row (line) contained within the L2G file (starting at zero).	MLSTART_COLUMN
The number of columns in each row (line) contained within the L2G file.	MLCOLS_PER_ROW
The start pixel of the first valid column in each row (line) contained within the L2G file (starting at zero).	MLSTART_PIX

Metadata Name/Description	M-API Constant
Number of observations per line	MLOBS_PER_LINE
SPSO_parameter	MLSPSO_PARAMETERS
Product type identifier: MOD09_ANG_L2G_1KM	M09ANG_PROD_ID
Zenith angle to sensor	M09SENSOR_ZENITH
Azimuth angle to sensor	M09SENSOR_AZIMUTH
Distance to sensor	M09SENSOR_DISTANCE
Zenith angle to sun	M09SOLAR_ZENITH
Azimuth angle to sun	M09SOLAR_AZIMUTH
Product type identifier: MOD09_PNT_L2G_1KM	M09PNT1K_PROD_ID
Product type identifier: MOD09_PNT_L2G_500M	M09PNT500_PROD_ID
Product type identifier: MOD09_PNT_L2G_250M i	M09PNT250_PROD_ID
Pointer to granule IDs from which the observation came. Zero relative. Fill value is 255.	M09GRANULE_PNT
Sample number of observation (1 km spatial element) in granule	M09OBS_IN_GRANULE
Sub-pixel (delta) line location of cell center in observation footprint. Relative to center of observation specified by (line, sample).	M09CELL_CENTER
Sub-pixel (delta) line location of cell center in observation footprint SDS. Relative to center of observation specified by (line, sample).	M09SAMPLE_CENTER
Observation coverage SDS: area of intersection between observation footprint and cell divided by area of observation.	M09OBS_COVERAGE
Cell coverage SDS: area of intersection between observation footprint and cell divided by area of cell.	M09CELL_COVERAGE
Product type identifier: MOD09_L2 and MOD13_L2	MOD09_L2G_500M M09_L2G_500M_PROD_ID
Surface Reflectance for MODIS Band 3	M09BAND3_SURF_REFL
Surface Reflectance for MODIS Band 4	M09BAND4_SURF_REFL
Surface Reflectance for MODIS Band 5	M09BAND5_SURF_REFL
Surface Reflectance for MODIS Band 6	M09BAND6_SURF_REFL
Surface Reflectance for MODIS Band 7	M09BAND7_SURF_REFL
Indicators of the quality of the 500 m reflectance data	M09QUALITY_500
Product type identifier: MOD09_L2 and MOD13_L2	M09_L2G_250M_PROD_ID
Surface Reflectance for MODIS Band 1	M09BAND1_SURF_REFL
Surface Reflectance for MODIS Band 2	M09BAND2_SURF_REFL
Indicators of the quality of the 250 m reflectance and VI data integrity.	M09QUALITY_250
Product type identifier: MOD09_L2 and MOD13_L2 MOD09SUBS_L2G_16DY	M09_REFLDB_PROD_ID
ang_size (in arcsec)	M09_REFLDB_ANGULAR_SIZE

Metadata Name/Description	M-API Constant
General information on observational basis M09_OBS_INFO words	M09_OBS_INFO_WORDS
Viewing and illumination angles	M09_ANGLES
N_obs_dy	M09_ANGLES_OBS
N_angles	M09_ANGLES_NUM
Surface reflectances	M09_REFLDB_SURF_REFL
N_obs_dy	M09_SURF_REFL_OBS
N_bands	M09_SURF_REFL_BANDS
Quality and weights of the respective observations	M09_QUALITY_WEIGHTS
N_obs_dy	M09_QUALITY_OBS
words	M09_QUALITY_WORDS
Product type identifier: MOD09_BARS	M09BARS_PROD_ID
Nadir-equivalent surface reflectances for MODIS bands 1-7	M09BARS
Overall quality of the BRDF-adjusted surface reflectances	M09BARS_QC
The number of columns in the full ISCCP grid for each row (line) contained within this L2G file.	M09NCOL
The start column in the full ISCCP grid for each row (line) contained within this L2G file (starting at zero).	M09ICOL_START
The number of columns in each row (line) contained within this L2G file.	M09NCOL_TILE
The start pixel of the first valid column in each row (line) contained within this L2G file (starting at zero).	M09IPIX_START
Product type identifier: MOD09_L3_16DY_G	M09_L3_PROD_ID
Identifier for BRDF models chosen	M09BRDF_MODEL_ID
RMSE for BRDF models chosen	M09BRDF_MODEL_RMSE
BRDF quality control	M09QUALITY
BRDF parameters for the seven land bands	M09BRDF_PARAMETERS
Albedo parameters for broadband, < 0.7 mu-m, > 0.7 mu-m, and the seven land bands.	M09ALBEDO
A neasure of fit from RMSE and sampling of all models tested.	M09FIT_ASSESS
The number of columns in the full ISCCP grid for each row (line) contained within this L3 file.	M09NCOL
The start column in the full ISCCP frid for each row (line) contained within this L3 file (starting at zero).	M09ICOL_START
The number of columns in each row (line) contained within this L3 file.	M09NCOL_TILE
The start pixel of the first valid column in each row (line) contained within this L3 file (starting at zero).	M09IPIX_START

Metadata Name/Description	M-API Constant
N_select_models	M09N_SELECT_MODELS
words	M09WORDS
land_bands	M09LAND_BANDS
number_parameters	M09NUMBER_PARAMETERS
land_bands_and_broadband_and_<>_0.7mu-m	M09LANDBANDS_BROADBAND_ OTHER
N_models	M09N_MODELS
Product type identifier: MOD09_L2 and MOD13_L2	M0913_L2_PROD_ID
SurfaceReflectance for MODIS Band 1 SDS	M0913BAND1_SURF_REFL
SurfaceReflectance for MODIS Band 2 SDS	M0913BAND2_SURF_REFL
SurfaceReflectance for MODIS Band 3 SDS	M0913BAND3_SURF_REFL
SurfaceReflectance for MODIS Band 4 SDS	M0913BAND4_SURF_REFL
SurfaceReflectance for MODIS Band 5 SDS	M0913BAND5_SURF_REFL
SurfaceReflectance for MODIS Band 6 SDS	M0913BAND6_SURF_REFL
SurfaceReflectance for MODIS Band 7 SDS	M0913BAND7_SURF_REFL
NDVI index at 250m	M0913_NDVI_INDEX
MVI index at 250m	M0913_MVI_INDEX
Indicators of the quality of the 250m reflectance and VI data integrity.	M0913QUALITY_250
Indicators of the quality of the 500m reflectance and VI data integrity.	M0913QUALITY_50
num_detectors	M0913NUM_DETECTORS
sampling	M0913SAMPLING
Number_of_pixels_processed	M10PROCESSED_PIXELS
Total_snow_pixels	M10SNOW_PIXELS
Percentage_snow	M10PERCENT_SNOW
Percentage_not_snow	M10PERCENT_NOT_SNOW
Above_range_NDSI	M10NDSI_ABOVE
Below_range_NDSI	M10NDSI_BELOW
Division_by_zero	M10ZERO_DIVIDE
Out_of_range_input	M10OUT_OF_RANGE_INPUT
No_decision	M10NO_DECISION
L2/L2G Identification of daily snow cover on the land surface	M10DAILY_SNOW
Product type identifier: MOD10_L2G	M10L2G_PROD_ID
Product type identifier: MOD10_L3_DY_G	M10L3_PROD_ID
L3 Identification of daily snow cover on the land surface	M10GRIDDED_SNOW
Product type identifier: MOD11_L2	M11L2_PROD_ID
L2/L2G Identification of Land Surface Temperature	M11SURF_TEMP
L2/L2G LST Quality Indicator	M11QUALITY
L2/L2G Error in land surface temperature measurements	M11ERRORS

Metadata Name/Description	M-API Constant
L2/L2G/L3 Band 31 emissivity	M11BAND31_EMIS
L2/L2G/L3 Band 32 emissivity	M11BAND32_EMIS
L2/L2G Band 29 or band 20 emissivity	M11BAND29OR20_EMIS
Product type identifier: MOD11_L2G	M11L2G_PROD_ID
Product type identifier: MOD11_L3_WK_G	M11L3_PROD_ID
L3 Identification of Land Surface Temperature	M11L3SURF_TEMP
Land surface temperature in view within 45deg	M11NARROW_LST
L3 LST Quality Indicator	M11L3QUALITY
Land-Surface Temperature Standard Deviation	M11STD_DEV
L3 Band 29 or band 20 emissivity	M11L3BAND29OR20_EMIS
Angular coefficients for Band 31 emissivity	M11BAND31_ANG_COEFS
Angular coefficients for Band 32 emissivity	M11BAND32_ANG_COEFS
Product type identifier: MOD12_L3_3MN_D/MOD12_L3_3MN_F	M12L3_PROD_ID
ang_size (in arcsec)	M12ANGULAR_SIZE
Identification of land cover type	M12LAND_COVER
Identification of Overall quality of the land cover	M12QUALITY
Identification of Number of products generated since last classification update	M12PRODS_GENERATED
Identification of Number of snow months over pervious 12 months	M12SNOW_MONTHS
Identification of Number of BRDFs used for classification that have been derived within the pass 12 month	M12BRDFS_USED
Identification of Confidence in BRDF/reflectance correction	M12BRDF_STOCK
Identification of Number of LST values used for classification	M12LST_VALS_USED
Identification of Confidence in VI over 12 months	M12VI_STOCK
Identification of TBD quality control for land cover type	M12QUALITY1
Identification of TBD quality control for land cover type	M12QUALITY2
Identification of Land cover change	M12LAND_COVER_CHANGE
Identification of Quality control for land cover change	M12CHANGE_QUALITY
Product type identifier: MOD14_L2	M14L2_PROD_ID
L2/L2G Identification of fire on the land surface	M14LAND_FIRE
L2/L2G/L3 Total emmitted energy detected	M14ENERGY
L2/L2G/L3 Class of fire detected	M14FIRE_CLASS
Fire quality control	M14QUALITY
Product type identifier: MOD14_L2G	M14L2G_PROD_ID
L2G/L3 Fire quality control	M14L2GQUALITY

Metadata Name/Description	M-API Constant
Product type identifier: MOD14_L3	M14L3_PROD_ID
Product type identifier: MOD29_L2	M29L2_PROD_ID
Total_sea_ice_pixels	M29SEA_ICE_PIXELS
Percentage_sea_ice	M29SEA_ICE_PERCENT
Percentage_not_sea_ice	M29NOT_SEA_ICE_PERCENT
Above_range_NDSI	M29NDSI_ABOVE
Below_range_NDSI	M29NDSI_BELOW
Division_by_zero	M29ZERO_DIVIDE
Out_of_range_input	M29OUT_OF_RANGE
No_decision	M29NO_DECISION
Identification of daily sea ice cover	M29DAILY_SEA_ICE
Product type identifier: MOD29_L2G	M29L2G_PROD_ID
Daily Ice Cover	M29L2GDAILY_SEA_ICE
Product type identifier: MOD29_L3_DY_G	M29L3_PROD_ID
Identification of daily sea ice cover	M29L3DAILY_SEA_ICE
Product type identifier: MOD33_L3_WK_G	M33L3_PROD_ID
Weekly Snow Cover	M33WEEKLY_SNOW
Product type identifier: MOD34_L3_MN	M34L3_PROD_ID
NDVI	M34NDVI
MVI	M34MVI
View zenith angles for NDVI	M34NDVI_ZENITH_ANGLES
View zenith angles for MVI	M34MVI_ZENITH_ANGLES
Quality control for NDVI	M34NDVI_QUALITY
Quality control for MVI	M34MVI_QUALITY
Product type identifier: MOD42_L3_WK_G	M42L3_PROD_ID
Weekly Sea Ice Cover	M42WEEKLY_SEA_ICE

APPENDIX C: DESCRIPTIONS AND PURPOSES

Appendix C shows the descriptions and purposes for both the C and FORTRAN routines. For a description of the variables refer to Appendix D and for a description of the associated error messages see Appendix E.

C.1 Descriptions and Purposes of C Routines

int **closeMODISfile** (MODFILE ***file*)

closeMODISfile terminates the access of M-API routines to a MODIS HDF file opened using **openMODISfile**. Only pre-existing files should be closed by **closeMODISfile**. **completeMODISfile** should be used to end access to a new MODIS HDF file so that the file's header information can be completed. **closeMODISfile** may fail to close the file if an error occurs.

```
int completeMODISfile (MODFILE **file, PGSt_MET_all_handles mdHandles,
                        ECSattr_names_for_all_handles HDFattrNames, long
                        int NumHandles)
```

completeMODISfile terminates the access of M-API routines to a MODIS HDF file opened using **openMODISfile**. In addition to closing the file, the file's standard header information is inserted. A pre-existing MODIS-HDF file should be closed by **closeMODISfile**or some of its header information will be over-written. **completeMODISfile** may fail to close the file if an error occurs.

See Chapter 4.5, Accessing Metadata, for a complete list of metadata **completeMODISfile** writes to the MODIS-HDF file before closing it.

int **createMODISarray** (MODFILE **file*, char **arrayname,* char **groupname,*
char **data_type,* long int *rank,*
long int *dimsizes*[])

createMODISarray creates an HDF SDS structure to store a new data array into a MODIS HDF file. It must be called before the data may be written to the file using **putMODISarray** or the attributes associated with the array may (optionally) be stored using **PMARIN** and **PMDMIN**.

The *groupname* string provides the facility to place the new array in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the array structure will not be created. The array may be placed in the file outside of any Vgroup by replacing *groupname* with NULL in C.

If an array with the name *arrayname* is written outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Arrays with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

`int` **`createMODISgroup`** `(MODFILE *`*`file`*`, char *`*`groupname,`* `char *`*`classname`*`)`

createMODISgroup creates an HDF Vgroup structure in a MODIS HDF file to store table and array structures. It must be called before any of the data objects to be aggregated in it are created. The use of data groups is optional, but data objects stored in them are documented in the MODIS Product File Definitions in Appendix F. A data group with the name *groupname* must be unique in a file. This prevents confusion that is caused by multiple data groups with the same name.

```
int createMODIStable (MODFILE *file, char *tablename, char *classname,
                      char *groupname, char*fieldname, char*data_type)
```

createMODIStable creates an HDF Vdata structure in a MODIS HDF file to store a new data table. It must be called before the data may be written to the file using **putMODIStable**. The text headers for each field (column) and the data type stored in each field must be provided.

The *groupname* string provides the facility to place the new table in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the table structure will not be created. The table may be placed in the file outside of any Vgroup by setting *groupname* = NULL in C.

If a table with the name *tablename* is created outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Tables with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

```
int getMODISardims (MODFILE *file, char *arrayname, char *groupname,
                    char *data_type, long int *rank, long int dimsizes[])
```

getMODISardims retrieves the essential characteristics of an HDF SDS array structure contained in a MODIS HDF file. This provides the information needed for properly reading data from the array structure using **getMODISarray**.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

Proper dimensioning of *dimsizes* to provide sufficient elements for the dimensions of the array structure may at first appear to require precognition. The easiest solution is to provide a generous (32 element) *dimsizes* array. Another approach is to use the *rank* variable as an input containing the number of elements in *dimsizes.* If *dimsizes* is inadequate for the multi-dimensional array structure in question, **getMODISardims** will fail gracefully but will return the rank of the array structure, allowing for the dimension information to be retrieved with a second call.

```
int getMODISarinfo  (MODFILE *file, char *arrayname, char *groupname, char
                    *attribute, char *data_type, long int *n_elements,
                    void *value)
```

getMODISarinfo retrieves the value stored in an HDF local attribute associated with an array structure given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the number of metadata values. getMODISarinfo replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *data_type* and *n_elements* are used to output information, these arguments may <u>not</u> be pointers to constants. GMARIN behaves similarly, so the arguments *nelmnt* and *dtype* must not be FORTRAN parameters or constants either.

n_elements, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **getMODISarinfo** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, **n_elements* is set to 0.

A variable of the proper data type and length should be passed for the *value* argument. The data type information required to properly use this routine may be found in Appendix F, Modis Data Product File Definitions.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if the argument*groupname* = NULL in C.

```
int getMODISarray (MODFILE *file, char *arrayname, char *groupname, long
                  int start[],  long int, dimsizes[], void *data)
```

getMODISarray returns a multi-dimensional array of data from an HDF SDS array structure contained in a MODIS HDF file. The data array must be of the same data type as data in the target array structure. In addition, the dimensions and array region requested from the array structure must be consistent with the structure's rank and dimensions. (The array structure's data type, rank, and dimensions may be retrieved using **getMODISardims**. If a **getMODISarray** error message occurs the data retrieval will not be performed. See Section 4.3, "Accessing Arrays" for additional information.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

```
int getMODISdiminfo (MODFILE *file, char *arrayname, char *groupname, long
                     int dimension, char *attribute, char *data_type,
                     long int *n_elements, void *value)
```

getMODISdiminfo retrieves the value stored in an HDF local attribute associated with an array structure's dimension given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the number of metadata values. getMODISdiminfo replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *data_type* and *n_elements* are used to output information, these arguments may not be pointers to constants. GMDMIN behaves similarly, so the arguments *nelmnt* and *dtype* must not be FORTRAN parameters or constants either.

n_elements, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **getMODISdiminfo** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, **n_elements* is set to 0.

A variable of the proper data type should be passed for the *value* argument. The data type information required to properly use either routine may be found in Appendix F, Modis Data Product File Definitions.

The *groupname* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if the argument*groupname* = NULL in C.

```
int getMODISECSinfo (MODFILE *file,  char *PVLAttrName, char *parmName,
                     char *data_type, long int *n_elements,  void *value)
```

getMODISECSinfo is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the MODIS Application Program Interface (API) Specification.

In HDF-EOS, parameters are collected together to form a text block using PVL. Then the text block is stored in HDF as a single attribute. **getMODISECSinfo** retrieve the value of a parameter from the PVL text block.

In order to obtain value of a parameter inside a PVL text block, the function reads the PVL text block specified by *PVLAttrName* from the MODIS file, creates the internal ODL tree structure from the PVL text block, and search the tree structure to retrieve the value of a parameter. The tree structure is then saved internally for consecutive searches in the same PVL text block for code efficiency. If multiple parameters will be retrieved from the same PVL block, just set *PVLAttrName* to the HDF PVL attribute name in the first call and set to NULL in C and ' ' in FORTRAN in the consecutive calls. If the next call is to retrieve the value of a parameter in a different PVL text block, set the PVLAttrName to the new PVL attribute name. The saved old tree structure will be deleted automatically and a new ODL tree will be created and saved. If you will no longer call getMODISECSinfo in your program and want to release the memory occupied by the saved tree, just set both *PVLAttrName* and *parmName* to NULL in C.

```
int getMODISfields (MODFILE *file,  char *tablename,  char *groupname, long
                    int *stringlen, long int *recno, long int *fieldno,
                    char *fieldname, char *data_type, char *classname)
```

getMODISfields retrieves the essential characteristics of an HDF Vdata table structure contained in a MODIS-HDF file. This provides the information needed for properly reading data from the table structure using **getMODIStable** or to write to it using **putMODIStable**. If any of the output parameters are set to NULL, then that data are not retrieved. An error (MFAIL) will be returned if 1) The output strings are not long enough to contain the data type or field name strings for all the Vdata's fields, 2) an unknown (e.g., not supported by the MODIS API) number type is encountered or 3) an HDF routine FAILs. The data type string (if requested) will be returned truncated to the point where the fault occurred.

stringlen, the address of the length of the *data_type* and *fieldname* output strings, is a required input if either of these strings is to be retrieved. getMODISfields normally replaces this input with the actual array length required to hold the larger ot the two output strings. If an unknown data type or an HDF routine fails, however, **stringlen* is set to 0.

The *groupname* string provides the facility to select a table structure existing in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tablename* if *groupname* = NULL in C.

```
int getMODISfileinfo (MODFILE *file, char *attribute, char *data_type,
                      long int *n_elements, void *value)
```

getMODISfileinfo retrieves the value associated with an attribute = value metadata pair given the attribute name. If the attribute cannot be found, the routine will return -1 and the passed variable unchanged.

The routine will also fail if the provided *data_type* is found to be different than the metadata's data type or the *n_elements* is found to be too small to contain the metadata's value. **getMODISfileinfo** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string). These metadata metadata may be used to properly retrieve the metadata value with a second call to the routine.

A variable of the proper data type should be passed for the *value* parameter. The data type information required to properly use either routine may be found in Appendix B, M-API-Supplied Constants, and Appendix F, MODIS Data Product File Definitions. Appendix B has a listing for each M-API provided metadata attribute that includes the data type, the format, and/or specific values associated with it.

```
int getMODIStable (MODFILE *file,  char *tablename, char *groupname,
                    char*fieldname, long int start, long int recno, long int
                    *buffsize, unsigned char *data)
```

getMODIStable retrieves one or more fields of data from one or more records in an HDF Vdata table structure contained in a MODIS-HDF file. The data are placed in the *data* buffer in consecutive records and in the order that the input *fieldnames* are listed. The length of this buffer must be able to contain all the fields requested times the number of records requested. If the *buffsize* input indicates that it is too small to contain the actual quantity of data requested, **getMODIStable** will fail, but it will return the actual *buffsize* required. The output *data* buffer must be at least this size. See Section 4.4, “Accessing Tables” for additional information.

The *groupname* string provides the facility to select a table structure placed in a particular HDF ‘Vgroup’ data group. Alternatively , the entire file will be searched for a table structure named *tablename* if groupname = NULL in C.

```
int putMODISarinfo (MODFILE *file, char *arrayname, char *groupname, char
                     *attribute, char *data_type, long int n_elements, void
                     *value)
```

putMODISarinfo attachs a local metadata attribute/value pair to a MODIS array. **putMODISarinfo** stores an attribute = value(s) metadata pair to the indicated array. If the attribute already exists, the value(s) will be updated.

```
int putMODISarray (MODFILE *file,  char *arrayname, char *groupname, long
                    int start[], long int, dimsizes[], void*data)
```

putMODISarray places a multi-dimensional array of data into an HDF SDS array structure previously created using **createMODISarray**. The data in the array must be of the data type the target array structure was created for. In addition, the dimensions and placement of the input array in the array structure must be consistent with the structure’s rank and dimensions. If a **putMODISarray** error message occurs, the data insertion will not be performed. See Section 4.3, “Accessing Arrays” for additional information. This routine may be called multiple times to fill the array structure. Data previously in the array structure may be overwritten.

The *groupname* string provides the facility to select an array structure placed in a particular HDF ‘Vgroup’ data group. The entire file will be searched for an array structure named *arrayname* if groupname = NULL in C.

```
int putMODISdiminfo (MODFILE *file, char *arrayname, char *groupname, long
                      int dimension, char *attribute, char *data_type, long
                      int n_elements, void *value)
```

putMODISdiminfo attachs a local attribute/value pair to a specific dimension of a MODIS array. **putMODISdiminfo** stores an attribute = value(s) attribute pair to the indicated dimension of a MODIS array. If the attribute already exists, the value(s) will be updated.

The *groupname* string provides the facility to select an array structure placed in a particular HDF ‘Vgroup’ data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (“ “) in FORTRAN.

```
int putMODISfileinfo (MODFILE *file, char *attribute, char *data_type,
                      long int n_elements, void * value)
```

putMODISfileinfo stores an attribute = value metadata pair to the indicated MODIS HDF file. If the attribute already exists, the value will be updated.

File attributes should be limited to M-API provided attribute macros. (See Section 5, M-API-Supplied Constants and Naming Conventions.) The data type should also be limited to the type associated with the MODIS file attribute, and the value itself restricted to that data type and the format and/or specific values associated with the attribute.

```
int putMODIStable (MODFILE *file, char *tablename, char *groupname, long
                   int start, long int recno, unsigned char *data)
```

putMODIStable places one or more data records into an HDF Vdata table structure previously created using **createMODIStable**. The data to be inserted into the table must be placed into a data array. The length of this array must be an integral number of the table structure's record length. The various data that make up a record should be inserted into the buffer in the same order as the field headers were ordered in the createMODIStable call. See Section 4.4, “Accessing Tables” for additional information. This routine may be called multiple times to fill the table structure. Data previously in the table structure may be overwritten.

The *groupname* string provides the facility to select a table structure placed in a particular HDF ‘Vgroup’ data group. The entire file will be searched for a table structure named *tablename* if *groupname* = NULL in.

```
int substrMODISECSinfo (char *char_value, long int n_elements, long int
                        *n_strings, char *substr[])
```

ECS metadata values may be integer, floating point, or character string values or arrays of values. Some may be multiple strings. The routine **getMODISECSinfo** retrieves such strings into a one-dimension character array with the individual strings separated by nulls (‘\0’). **substrMODISECSinfo** breaks this ‘packed’ character array into its constituent substrings. **substrMODISECSinfo** sets the pointers in a provided output array to the beginning of each substring in the *char_value* array.

```
int32 searchMODISgroup (MODFILE *file,  char *groupname, char *classname,
                        char *objectname, char *objectclass, int32
                        objecttype)
```

searchMODISgroup searches an HDF Vgroup structure in a MODIS HDF file to find if an HDF object is in the Vgroup. Both the group and the object are specified by their name and class name. However, the classname is an optional feature. If class names are set to NULL, only name comparison is performed. Because SDS (array) has no class name, the objectclass for an SDS is always ignored. If the specified object exists, the function will return the reference id for Vdata and Vgroup, and index for SDS. If the object does not exist, the function will return NO_OBJECT, which is defined in mapic.h as -2.

The *groupname* string provides the facility to select an array structure placed in a particular HDF ‘Vgroup’ data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (“ “) in FORTRAN.

`long int` **`MODISsizeof`** `(char` *`*data_type`*`)`

The M-API uses a set of standard strings to describe the data types in stored in array and table structures. These strings are returned, for example, by the routine **getMODISardims** to describe the data type of the target array structure. **MODISsizeof** returns the number of bytes required to store a data type given this data type string. The input string may be a series of comma-delimited data type strings, in which case the total number of bytes to store the record described by the string is returned.

`MODFILE *` **`openMODISfile`** `(char` *`*filename`*`, char` *`*access`*`)`

openMODISfile opens an HDF file and creates the HDF structures to support the M-API routines access to it. openMODISfile must be called to produce the MODFILE structure before any of these routine can access it. Note that setting the file access to "w" creates a file and will overwrite a pre-existing one. Will close the file and return null outputs if an error occurs.

C.2 Descriptions and Purposes of FORTRAN Routines

`INTEGER FUNCTION` **`CLMFIL`** `(`*`modfil`*`)`

CLMFIL terminates the access of M-API routines to a MODIS HDF file opened using **OPMFIL**. Only pre-existing files should be closed by closeMODISfile. **CPMFIL** should be used to end access to a new MODIS HDF file so that the file's header information can be completed. **CLMFIL** may fail to close the file if an error occurs.

`INTEGER FUNCTION` **`CPMFIL`** *`(modfil, mdhandle, hdfattrnms, numhands)`*

CPMFIL terminates the access of M-API routines to a MODIS HDF file created using **OPMFIL**. In addition to closing the file, the MODIS file's standard header information is inserted. A pre-existing MODIS HDF file should be closed by **CLMFIL** or some of its header information will be overwritten. **CLMFIL** may fail to close the file if an error occurs.

See Section 4.5, Accessing Metadata, for a complete list of metadata completeMODISfile writes to the MODIS HDF file before closing it.

`INTEGER FUNCTION` **`CRMAR`** `(`*`modfil, arrnm, grpnm, dtype, rank, dims`*`)`

CRMAR creates an HDF SDS structure to store a new data array into a MODIS HDF file. It must be called before the data may be written to the file using **PMAR** or the attributes associated with the array may (optionally) be stored using **PMARIN** and **PMDMIN**.

The *grpnm* string provides the facility to place the new array in an HDF 'Vgroup' data group. If a Vgroup with the name *groupname* does not exist, the array structure will not be created. The array may be placed in the file outside of any Vgroup by replacing *grpnm* = a blank string (' ') in FORTRAN.

If an array with the name *arrnm* is written outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Arrays with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

`INTEGER FUNCTION` **`CRMGRP`** `(`*`modfil, grpnm, clsnm`*`)`

CRMGRP is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the M-API Specification.

CRMGRP creates an HDF Vgroup structure in a MODIS HDF file to store table and array structures. It must be called before any of the data objects to be aggregated in it are created. The use of data groups is optional, but data objects stored in them are documented in the MODIS Product File Definitions in Appendix F. A data group with the name *grpnm* must be unique in a file. This prevents confusion that is caused by multiple data groups with the same name.

`INTEGER FUNCTION` **`CRMTBL`** `(`*`modfil, tblnm, clsnm, grpnm, fldnm, dtype`*`)`

CRMTBL creates an HDF Vdata structure in a MODIS HDF file to store a new data table. It must be called before the data may be written to the file using **PMTBL**. The text headers for each field (column) and the data type stored in each field must be provided.

The *grpnm* string provides the facility to place the new table in an HDF 'Vgroup' data group. If a Vgroup with the name *grpnm* does not exist, the table structure will not be created. The table may be placed in the file outside of any Vgroup by setting *grpnm* = ' ' in FORTRAN.

If a table with the name *tblnm* is created outside of a Vgroup, it must not already exist in the file. This is to prevent the confusion caused by multiple data objects with the same name. Tables with the same name may be stored in the same file, however, if they are placed in separate Vgroups.

`INTEGER FUNCTION` **`GMAR`** `(`*`modfil, arrnm, grpnm, start, dims, data)`*

GMAR returns a multi-dimensional array of data from an HDF SDS array structure contained in a MODIS HDF file. The data array must be of the same data type as data in the target array structure. In addition, the dimensions and array region requested from the array structure must be consistent with the structure's rank and dimensions. (The array structure's data type, rank, and dimensions may be retrieved using **GMARDM**). If a **GMAR** error message occurs the data retrieval will not be performed. See Section 4.3, "Accessing Arrays" for additional information.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (' ') in FORTRAN.

`INTEGER FUNCTION` **`GMARDM`** `(`*`modfil, arrnm, grpnm, dtype, rank, dims)`*

GMARDM retrieves the essential characteristics of an HDF SDS array structure contained in a MODIS HDF file. This provides the information needed for properly reading data from the array structure using **GMAR**.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (" ") in FORTRAN.

Proper dimensioning of *dims* to provide sufficient elements for the dimensions of the array structure may at first appear to require precognition. The easiest solution is to provide a generous (32 element) *dims* array. Another approach is to use the *rank* variable as an input containing the number of elements in *dims.* If *dims* is inadequate for the multi-dimensional array structure in question, **GMARDM** will fail gracefully but will return the rank of the array structure, allowing for the dimension information to be retrieved with a second call.

`INTEGER FUNCTION` **`GMARIN`** `(`*`modfil,arrnm,grpnm,attrib, dtype,nelmnt,value`*`)`

GMARIN retrieves the value stored in an HDF local attribute associated with an array structure given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the number of metadata values. **GMARIN** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *dtype* and *nelmnt* are used to output information, these arguments may <u>not</u> be pointers to constants.

nelmnt, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **GMARIN** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, *nelmnt* is set to 0.

A variable of the proper data type and length should be passed for the *value* argument. The data type information required to properly use this routine may be found in Appendix F, Modis Data Product File Definitions.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument*grpnm* = *grpnm* is a blank string (" ") in FORTRAN.

`INTEGER FUNCTION` **`GMDMIN`** `(`*`modfil, arrnm, grpnm, dim, attrib, dtype, nelmnt, value`*`)`

GMDMIN retrieves the value stored in an HDF local attribute associated with an array structure's dimension given the attribute name. If the attribute cannot be found, the routine will return MFAIL (-1) .

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the number of metadata values. getMODISdiminfo replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string, including the '\0' terminator). The retrieved data type and attribute array size information may then be used to properly retrieve the array structure metadata with a second call to the routine. Since *dtype* and *nelmnt* are used to output information, these arguments may <u>not</u> be pointers to constants.

nelmnt, the address of the number of elements in the provided output *value* array, is a required input if the metadata are to be retrieved. **GMDMIN** normally replaces this input with the actual array length required to hold this metadata. If the local attribute is not found or an HDF routine fails, however, *nelmnt* is set to 0.

A variable of the proper data type should be passed for the *value* argument. The data type information required to properly use either routine may be found Appendix F, Modis Data Product File Definitions.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument*grpnm* = a blank string (" ") in FORTRAN.

INTEGER FUNCTION **GMECIN** (*modfil, pvlname, pname, nms, dtype, pvalue*)

GMECIN is part of a larger software system called the MODIS Applications Programming Interface (API) Utility, abbreviated M-API. The M-API Utility consists of subroutines which allow MODIS Science Team-supplied software to read and write data and metadata from/to HDF files. The functionality of the M-API is defined in the M-API Specification.

In HDF-EOS, parameters are collected together to form a text block using PVL. Then the text block is stored in HDF as a single attribute. GMECIN retrieve the value of a parameter from the PVL text block.

In order to obtain value of a parameter inside a PVL text block, the function reads the PVL text block specified by *pvlname* from the MODIS file, creates the internal ODL tree structure from the PVL text block, and search the tree structure to retrieve the value of a parameter. The tree structure is then saved internally for consecutive searches in the same PVL text block for code efficiency. If multiple parameters will be retrieved from the same PVL block, just set pvlname to the HDF PVL attribute name in the first call and set to ' ' in the consecutive calls. If the next call is to retrieve the value of a parameter in a different PVL text block, set the pvlname to the new PVL attribute name. The saved old tree structure will be deleted automatically and a new ODL tree will be created and saved. If you will no longer call **GMECIN** in your program and want to release the memory occupied by the saved tree, just set both *pvlname* and *pname* to ' ' .

INTEGER FUNCTION **GMFIN** (*modfil, attrib, dtype, nelmnt, value*)

GMFIN retrieves the value associated with an attribute = value metadata pair given the attribute name. If the attribute cannot be found, the routine will return -1 and the passed variable unchanged.

The routine will also fail if the provided *dtype* is found to be different than the metadata's data type or the *nelmnt* is found to be too small to contain the metadata's value. **GMFIN** replaces this input information with the actual data type and number of elements contained in the metadata value (in the case of character data, it is the length of the string). These metadata metadata may be used to properly retrieve the metadata value with a second call to the routine.

A variable of the proper data type should be passed for the *value* parameter. The data type information required to properly use either routine may be found in Appendix B, M-API-Supplied Constants, and Appendix F, MODIS Data Product File Definitions. Appendix B has a listing for each M-API provided metadata attribute that includes the data type, the format, and/or specific values associated with it.

`INTEGER FUNCTION` **`GMFLDS`** `(`*`modfil, tblnm, grpnm, strln, recno, fldno, fldnm, dtype, clsnm`*`)`

GMFLDS retrieves the essential characteristics of an HDF Vdata table structure contained in a MODIS-HDF file. This provides the information needed for properly reading data from the table structure using **GMTBL** or to write to it using **PMTBL**. If any of the output parameters are set to NULL, then that data are not retrieved. An error (MFAIL) will be returned if:

1) The output strings are not long enough to contain the data type or field name strings for all the Vdata's fields,

2) an unknown (e.g., not supported by the MODIS API) number type is encountered or

3) an HDF routine FAILs. The data type string (if requested) will be returned truncated to the point where the fault occurred.

stringlen, the address of the length of the *dtype* and *fname* output strings, is a required input if either of these strings is to be retrieved. **GMFLDS** normally replaces this input with the actual array length required to hold the larger ot the two output strings. If an unknown data type or an HDF routine fails, however, **stringlen* is set to 0.

The *grpnm* string provides the facility to select a table structure existing in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tblnm* if *grpnm* = a blank string (' ') in FORTRAN.

`INTEGER FUNCTION` **`GMTBL`** `(`*`modfil, tblnm, grpnm, fldnm, start, recno, buffsz, data)`*

GMTBL retrieves one or more fields of data from one or more records in an HDF Vdata table structure contained in a MODIS-HDF file. The data are placed in the *data* buffer in consecutive records and in the order that the input *fldnm* are listed. The length of this buffer must be able to contain all the fields requested times the number of records requested. If the *buffsz* input indicates that it is too small to contain the actual quantity of data requested, **GMTBL** will fail, but it will return the actual *buffsz* required. The output *data* buffer must be at least this size. See Section 4.4, "Accessing Tables" for additional information.

The *grpnm* string provides the facility to select a table structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for a table structure named *tblnm* if *grpnm* = a blank string (' ') in FORTRAN.

`INTEGER FUNCTION` **`MSIZE`** `(`*`dtype`*`)`

The M-API uses a set of standard strings to describe the data types in stored in array and table structures. These strings are returned, for example, by the routine **GMARDM** to describe the data type of the target array structure. **MSIZE** returns the number of bytes required to store a data type given this data type string. The input string may be a series of comma-delimited data type strings, in which case the total number of bytes to store the record described by the string is returned.

`INTEGER FUNCTION` **`OPMFIL`** `(`*`fname`*`,` *`access`*`,` *`modfil`*`)`

OPMFIL opens an HDF file and creates the HDF structures to support the M-API routines access to it. **OPMFIL** must be called to produce the FORTRAN modfil array before any of these routine can access it. Note that setting the file access to "w" creates a file and will overwrite a pre-existing one. **OPMFIL** will close the file and return null outputs if an error occurs.

`INTEGER FUNCTION` **`PMAR`** `(`*`modfil, arrnm, grpnm, start, dims, data)`*

PMAR places a multi-dimensional array of data into an HDF SDS array structure previously created using **CRMAR**. The data in the array must be of the data type the target array structure was created for. In addition, the dimensions and placement of the input array in the array structure must be consistent with the structure's rank and dimensions. If a **PMAR** error message occurs, the data insertion will not be performed. See Section 4.3, "Accessing Arrays" for additional information. This routine may be called multiple times to fill the array structure. Data previously in the array structure may be overwritten.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. The entire file will be searched for an array structure named *arrnm* if *grpnm* = a blank string (" ") in FORTRAN.

`INTEGER FUNCTION` **`PMARIN`** `(`*`modfil, arrnm, grpnm, dtype, nelmnt, value)`*

PMARIN stores an attribute = value metadata pair in an HDF local attribute associated with an array. The SDS array structure must be created (using **CRMAR**) prior to attaching a dimension attribute to it If the attribute already exists, the value(s) are updated.

The *grpnm e* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively, the entire file will be searched for an array structure named *arrnm* if the argument *grpnm* = NULL in C.

`INTEGER FUNCTION` **`PMDMIN`** `(`*`modfil, arrnm, grpnm, dtype, nelmnt, value)`*

PMDMIN stores an attribute = value metadata pair in an HDF local attribute associated with an array structure's dimension. The SDS array structure must be created (using **CRMAR**) prior to attaching a dimension attribute to it. If the attribute already exists, the value(s) are updated.

The *grpnm* string provides the facility to select an array structure placed in a particular HDF 'Vgroup' data group. Alternatively , the entire file will be searched for an array structure named *arrnm* if the argument *grpnm e* = NULL in C.

`INTEGER FUNCTION` **`PMFIN`** `(`*`modfil, attrib, dtype, nelmnt, value`*`)`

PMFIN stores an attribute = value metadata pair to the indicated MODIS HDF file. If the attribute already exists, the value will be updated.

File attributes should be limited to M-API provided attribute macros. (See Section 5, M-API-Supplied Constants and Naming Conventions.) The data type should also be limited to the type associated with the MODIS file attribute, and the value itself restricted to that data type and the format and/or specific values associated with the attribute.

INTEGER FUNCTION **PMTBL** (*modfil, tblnm, grpnm, start, recno, datasz, data)*

PMTBL places one or more data records into an HDF Vdata table structure previously created using **CRMTBL**. The data to be inserted into the table must be placed into a data array. The length of this array must be an integral number of the table structure's record length. The various data that make up a record should be inserted into the buffer in the same order as the field headers were ordered in the **CRMTBL** call. See Section 4.4, “Accessing Tables” for additional information. This routine may be called multiple times to fill the table structure. Data previously in the table structure may be overwritten.

The *grpnm* string provides the facility to select a table structure placed in a particular HDF ‘Vgroup’ data group. The entire file will be searched for a table structure named *tblnm* if *grpnm* = ‘ ’ in FORTRAN.

INTEGER FUNCTION **SMECIN** (*cvalue*, *nelmnt*, *nstrs*, *substr*)

ECS metadata values may be integer, floating point, or character string values or arrays of values. Some may be multiple strings. The routine **GMECIN** retrieves such strings into a one-dimension character array with the individual strings separated by nulls ('\0'). SMECIN breaks this 'packed' character array into its constituent *substr*ings. **SMECIN** copies these *substr*ings into separate rows of a FORTRAN character string array.

INTEGER FUNCTION **SRMGRP** (modfil, grpnm, clsnm, objnm, objcls, objtyp)

SRMGRP searches an HDF Vgroup structure in a MODIS HDF file to find if an HDF object is in the Vgroup. Both the group and the object are specified by their name and class name. However, the classname is an optional feature. If class names are set to NULL, only name comparison is performed. Because SDS (array) has no class name, the objectclass for an SDS is always ignored. If the specified object exists, the function will return the reference id for Vdata and Vgroup, and index for SDS. If the object does not exist, the function will return NO_OBJECT. The NO_OBJECT is defined in mapic.inc as -2.

The *groupname* string provides the facility to select an array structure placed in a particular HDF ‘Vgroup’ data group. Alternatively , the entire file will be searched for an array structure named *arrayname* if groupname = NULL in C and a blank string (“ “) in FORTRAN.

(This page intentionally left blank.)

APPENDIX D: VARIABLES FOR ROUTINES

Table D-1 Variables for C Routines

Parameter	Data Type	Definition
access	char *	IN: Standard C access mode. One of: "r" Open for read only. "w" Create for read/write, over writes pre-existing files. "a" Open for read/write, creates a file that doesn't exist.
arrayname	char *	IN: ASCII string that will be the name of the array, up to 256 characters long. Array names cannot begin with a blank character and trailing blanks should be removed or else FORTRAN programs will have difficulty accessing them.
attribute	char *	IN: ASCII string name of the attribute. Provided macros for accepted MODIS HDF file attribute names are listed in Appendix B, M-API-Supplied Constants.
buffsize	long int *	IN/OUT: Address of the *data* buffer size on input, in bytes. The buffer must be at least this size. *buffsize* will normally return the number of bytes of data successfully retrieved. If the buffer is too small, however, the routine returns MFAIL and *buffsize* will contain the size a buffer must be to contain the output data. If a functional error occurs, it is set to 0 because making this output size determination will be unreliable.
char_value	char *	IN: Character string containing the 'packed' multiple substrings of ECS metadata retrieved with getMODISECSinfo. Do not deallocate *char_value* until *substr* array gets correct values.
classname	char *	IN: ASCII string that will be the class name of the table, up to 64 characters long. If set to NULL or an empty string, the table will have no class. OUT: ASCII string for the class name of the table structure. Provided array may be up to 64 bytes long.
data	void * and unsigned char *	IN/OUT: Address of the data buffer.

Parameter	Data Type	Definition

data_type	char *	IN/OUT: Address of the data type of the *value* output. The attribute's value will not be retrieved unless the input data type matches that of the attribute.
		NOTE: This argument must not be a the address of a constant string and should point to memory at least 8 bytes long.
		Permitted C data types:
		`"int8"`
		`"uint8"`
		`"int16"`
		`"uint16"`
		`"int32"`
		`"uint32"`
		`"float32"`
		`"float64"`
		`"char *"`

dimension	long int *	IN: The dimension number which the attribute is attached to (0-based). getMODISdiminfo associates the 0 dimension with the <u>least</u> rapidly varying array index of an HDF SDS array structure.
dimsizes	long int *	IN: The size of the array being retrieved from the array structure. The *dimsize* array must have the same number of elements as the target array structure has dimensions and the product of the array dimensions must equal the number of elements in *data.* OUT: Array describing the size of each dimension of the target HDF array structure*.* The dimensions will not be provided unless dimsizes contains sufficient elements for the rank of the array.
ECSattr_names_for_all_handles	long int *	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS][MAX_ECS_NAME_L], where PGSd_MET_NUM_OF_GROUPS] is 20 and MAX_ECS_NAME_L is 50. This array is typedef-ed as ECSattr_names_for all_handles. Each row in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in mdHandles array. Each name, which is a string, should be less that MAX_ECS_NAME_L characters and occupies one row in the array. Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.
fieldname	char *	IN: Array of comma-delimited ASCII string table headers. The headers should be in the same order that the data for each table row will subsequently be written in. Each field name must be less than 128 characters long and the Vdata table may contain up to 36 fields. OUT: Array of comma-delimited ASCII string table headers.

Parameter	**Data Type**	**Definition**
fieldno	long int *	OUT: Number of fields (columns) present in the table structure.
file	Modfile*	IN/OUT: Pointer to MODFILE structure address used to reference a file in all M-API routines. Set to NULL when the file is successfully closed.
filename	char *	IN: Path and filename for the file to be opened, up to 255 characters long.
groupname	char *	IN: ASCII string name of the data group containing the target array structure. For ‘GET’ functions: If set to NULL the entire file will be searched for the array structure named *arrayname.* For ‘PUT’ functions: If set to NULL or an empty string, the table will not be placed in a data group.
HDFattrNames	Modfile*	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [MAX_ECS_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and MAX_ECS_NAME_L is 256. This array is typedef-ed as *ECSattr_names_for_all_handles*. Each row in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in *mdHandles* array. Each name, which is a string, should be less than MAX_ECS_NAME_L characters and occupies one row in the array.
mdHandles	Modfile*	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [PGSd_MET_GROUP_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as **PGSt_MET_all_handles**. Each row in the array stores a handle to an internal ODL tree structure which will be written out as an ECS PVL attribute. Each handle, which is a string, should be less than 50 characters and occupy one row in the array. Therefore, the maximum number of handles should be 20.
n_elements	long int *	IN/OUT: Address of the number of memory elements as *data_type* available in the *value* array. The attribute's value will not be retrieved unless **n_elements* indicates that there is sufficient space available in *value*. getMODISECSinfo replaces this input with the number of elements required to contain the metadata. If the parameter cannot be found, **n_element* will be left unchanged, or set to 0 if a function error occurs. NOTE: This argument must not be the address of a constant.

Parameter	Data Type	Definition

n_elements (continued)	long int *	**SPECIAL CASE for multiple strings:** If there are multiple character strings for the parameters, strings will be packed together and returned in *value* . The separator between strings is '\0'. The low 16 bit of *n_elements* will return the total bytes in the values, including the '\0's between the strings and the '\0' at the end of last string. The part above the low 16 bits will return number of strings packed - 1. To obtain how many string retrieved, do the calculation:
		n_strings = *n_elemets/65536 + 1
		n_bytes = *n_elements%65536
		Therefore, if **n_elements* is less than 65536, there is only one strings in *value* and **n_elements* is the number of bytes (characters) in the string, including the last '\0'.

n_strings	long int *	IN/OUT: Address of the number of pointers available in the *substr* array. The *substr* pointers will not be set to the substrings in *char_value* unless there are sufficient pointers available in the pointer array. substrMODISECSinfo replaces this input with the number of substrings pointers have been set to in the *char_value* array. **n_strings* will be set to 0 if a function error occurs. This argument must not be the address of a constant.
NumHandles	long int *	IN: Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.
objectclass	char *	IN: (Optional)ASCII string of the class name of the data object. Set to NULL for not comparing the object class
objectname	char *	IN: ASCII string of the object name to be searched.
objecttype	int32	IN: Type of the object; The valid objects are: DFTAG_NDG (for SDS) DFTAG_VH (for Vdata, or attribute if the object class is set to Attr0.0) DFTAG_VG (for Vgroup).
parmName	char *	IN: ASCII string name of a parameter whose value will be retrieved. Set both PVLAttrName and parmName to NULL in C will release the memory occupied by the internal ODL tree. The parmName could parameter name only or combination of name and class represented as "name.class".
PGSt_MET_all_handles	char *	IN: A character array with size of [pGSd_MET_NUM_OF_GROUPS][PGS_MET_GROUP_NAME_L], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as PGSt_MET_all_handles. Each row in the array stores a handles to an internal ODL tree structure which will be written out a an ECS PVL attribute. Each handles, which is a string, should be less than 50 characters and occupy, one row in the array. Therefore, the maximum number of handles should be 20.

Parameter	Data Type	Definition
PVLAttrName	char *	IN: ASCII string name of the HDF attribute which contains the PVL text block. Set PVLAttrName to NULL in C while parmName is not NULL in C or not ‘ ’ in FORTRAN will result in searching the last PVL text block for the value of *parmName* parameter.
rank	long int *	IN/OUT: The number of elements in the array *dimsizes* on input. This is replaced with the number of dimensions in the target HDF array structure for output. It is set to 0 if a functional error occurs. No dimensional information will be provided if rank = NULL.
recno	long int *	IN: Number of records being inserted into the table structure. OUT: Number of records(rows) present in the table structure. NOTE: The product of *recno* and the table structure's record length must have the same length as the buffer addressed by *data*
reproc_status	char *	IN: Intent to reprocess the data.
start	long int *	IN: Zero-based record location to begin placing reading the data into the table structure. NOTE: The *start* array must have the same number of elements as the target array structure has dimensions. The *start* location must be contiguous to the location of records already in the table. For placing If *start* = -1 data records will be appended to the end of the table structure.
stringlen	long int *	IN/OUT: Input of the minimum length of *fieldname* and *data_type* arrays. Returns the minimum array length actually required to hold the longer of the two strings. It is set to 0 if a functional error occurs.
substr	char *	OUT: Array of poiners to the constituent substrings contained in the *char_value* array.
tablename	char *	IN: ASCII string that will be the name of the table, up to 64 characters long. Table names should not include trailing blanks or else FORTRAN programs will have difficulty accessing them.
temporal_coverage	char *	IN: Description observation period in ECS metadata syntax.
value	void	IN: Address of the data to store in the in the attribute. If the attribute already exists, the value will be updated. Values should conform to the data types, formats and/or those values enumerated for the attribute in Appendix B, M-API-Supplied Constants. OUT: Buffer for the value. User should allocate enough memory for this buffer. If there are multiple data values in character type, the value will be placed consecutively. If the data value type is "char *", string will be separated by '\0'.

Table D-2 Variables for FORTRAN Routines

Parameter	Data Type	Definition
access	Character*(*)	IN: Standard C access mode. One of: 'r' Open for read only. 'w' Create for read/write. 'a' Open for read/write (append.).
arrnm	Character*(*)	IN: ASCII string name of the target HDF array structure, up to 128 characters long. Array names cannot begin with a blank character.
attrib	Character*(*)	IN: ASCII string name of the attribute. Provided macros for accepted MODIS HDF file attribute names are listed in Appendix B, M-API-Supplied Constants.
buffsize	Integer	IN/OUT: The *data* buffer size on input, in bytes. The buffer must be at least this size. *buffsize* will normally return the number of bytes of data successfully retrieved. If the buffer is too small, however, the routine returns MFAIL and *buffsize* will contain the size a buffer must be to contain the output data requested. If a functional error occurs, it is set to 0 because making this output size determination will be unreliable.
clsnm	Character*(*)	IN: ASCII string that will be the class name of the table, up to 64 characters long. If set to a blank string, the table will have no class. OUT: ASCII string for the class name of the table structure. Provided array should be at least (64) bytes long.
cvalue	Character*(*)	IN: Character string containing the 'packed' multiple substrings of ECS metadata retrieved with GMECIN.
data	<any>	IN/OUT: Multi-dimensional data array. NOTE:
dim	Integer	IN: The dimension number which the attribute is attached to (0-based). GMDMIN associates the 0 dimension with the <u>most</u> rapidly varying array index of an HDF SDS array structure.
dims	Integer	IN: The size of the array being inserted into the array structure. The *dims* array must have the same number of elements as the target array structure has dimensions and the product of the array dimensions must equal the number of elements in *data.* OUT: Array describing the size of each dimension of the target HDF array structure. The dimensions will not be provided unless dims contains sufficient elements for the rank of the array. (HDF 3.3r4 SDS's may contain up to 32 dimensions.)

Parameter	Data Type	Definition

dtype	Character*(*)	IN/OUT: Data type of the *value* output. The attribute’s value will not be retrieved unless the input data type matches that of the attribute. GMARIN replaces with the data type of the retrieved metadata.
		NOTE: This argument must not be a parameter or constant. The memory size of *dtype* should be at least 13 characters long.
		Permitted FORTRAN data types:
		`‘INTEGER*1’`
		`‘UINTEGER*1’`
		`‘INTEGER*2’`
		`‘UINTEGER*2’`
		`‘INTEGER*4’`
		`‘UINTEGER*4’`
		`‘REAL*4’`
		`‘REAL*8’`
		`‘CHARACTER*(*)’`

fldnm	Character*(*)	IN: Array of comma-delimited ASCII string table headers. The headers should be in the same order that the data for each table row will subsequently be written in. Each field name must be less than 128 characters long and the Vdata table may contain up to 36 fields. OUT: Array of comma-delimited ASCII string table headers.
fldno	Integer	OUT: Number of fields (columns) present in the table structure.
fname	Character*(*)	IN/OUT: Number of elements availabel in the *value* array. Output replaces with the number of elements required to contain the metadata.
grpnm	Character*(*)	IN: ASCII string name of the data group containing the target array structure. OUT: ASCII string name of the data group to place the new array in. For ‘GET’ functions: If *grpnm* = ’ ’ the entire file will be searched for the array structure named *arrnm/tblnm.* For ‘PUT’ functions: If set to “ “(blank) the array will not be placed in a data group.
hdfatrnms	Character*255(*)	IN: A character array with size of [PGSd_MET_NUM_OF_GROUPS] [MAX_ECS_NAME_L-1], where PGSd_MET_NUM_OF_GROUPS is 20 and MAX_ECS_NAME_L is 256. Each string in this array is a character string used as a global attribute name for storing an ECS PVL text block which has a handle in the corresponding row in mdHandles array. Each name, which is a string, should be less that MAX_ECS_NAME_L characters and occupies one row in the array.

Parameter	Data Type	Definition
mdhandle	Character*45(*)	IN: An array of character strings. The memory size of the array is [PGSd_MET_NUM_OF_GROUPS] [PGS_MET_GROUP_NAME_L-1], where PGSd_MET_NUM_OF_GROUPS is 20 and PGSd_MET_GROUP_NAME_L is 50. This array is typedef-ed as PGSt_MET_all_handles. Each row in the array stores a handles to an internal ODL tree structure which will be written out a an ECS PVL attribute. Each handles, which is a string, should be less than 50 characters and occupy, one row in the array. Therefore, the maximum number of handles should be 20.
modfil	Integer	IN: Array that is used to reference a MODIS HDF file in all other M-API routines. OUT: Array that is used to reference the file in all other M-API routines. The array will return all zeroes if an error occurs.
nelmnt	Integer	IN: The composite output dimensions, from GMECIN, containing (in the case of character string metadata the total length (in bytes) of the string in *cvalue* in its lower two bytes and the number of substrings packed into *cvalue* less one in the upper two bytes. The calculations: n_strings = n_elements/65536 + 1 n_bytes = n_elements%65536 provide the number of substrings and the total length, respectively, of the data in *cvalue*. When there is only one string in *cvalue*, *nelmnt* will be less than 65536 and there is no need to use SMECIN. OUT: Number of elements availabel in the *value* array. The attribute's value will not be retrieved unless *nelmnt* indicates that there is sufficient space in *value*. Output replaces with the number of elements required to contain the metadata. If a function error occurs, however, *nelmnt* is set to 0. This argument must not be a parameter or constant.
nms	Character*(*)	IN/OUT: The number of memory elements as *dtype* available in the *value* array. The attribute's value will not be retrieved unless *nms* indicates that there is sufficient space available in *value*. GMECIN replaces this input with the number of elements required to contain the metadata. If the parameter cannot be found, **nms* will be left unchanged, or set to 0 if a function error occurs. This argument must be a variable.

Parameter	Data Type	Definition
nms (continued)	Character*(*)	**SPECIAL CASE for multiple strings:** If there are multiple character strings for the parameters, strings will be packed together and returned in *value* . The separator between strings is '\0' (numerical value 0). The low 16 bit of *nms* will return the total bytes in *value*, including the '\0's. The part above the low 16 bits will return (number of strings packed - 1). To obtain how many string retrieved, do the calculation: n_strings = nms/65536 + 1 n_bytes = MOD(nms, 65536) Therefore, if *nms* is less than 65536, there is only one strings in *value* and *nms* is the number of bytes (characters) in the string.
nstrs	Integer	IN/OUT: Number of elements available in the *substr* array. The *substr* will not be set to the substrings in *cvalue* unless there are sufficient elements available in the *substr* array. SMECIN replaces this input with the number of substrings already set in the *cvalue* array. *nstrs* will be set to 0 if a function error occurs.
numhands	Integer	IN: Specifies the number of actual handles contained in mdHandles. This may be set from 0 to PGSd_MET_NUM_OF_GROUPS.
objcls	Character*(*)	IN: (Optional)ASCII string of the class name of the data object. Set to NULL for not comparing the object class.
objnm	Character*(*)	IN: ASCII string of the object name to be searched.
objtyp	Integer	IN: type of the object; The valid objects are: DFTAG_NDG, DFTAG_VH, DFTAG_VG.
pname	Character*(*)	IN: ASCII string name of a parameter whose value will be retrieved. Set both *pvlname* and *pname* to ' ' will release the memory occupied by the internal ODL tree. The pname could parameter name only or combination of name and class represented as "name.class".
pvlname	Character*(*)	IN: ASCII string name of the HDF attribute which contains the PVL text block. Set *pvlname* to ' ' while pname is not equal to ' ' will result in searching the last PVL text block for the value of *pname* parameter.
rank	Integer	IN/OUT: The number of elements in the array *dimsizes* on input. This is replaced with the number of dimensions in the target HDF array structure for output. It is set to 0 if a functional error occurs.
recno	Integer	IN: Number of records being inserted into the table structure. The product of *recno* and the table structure's record length must have the same length as the buffer addressed by *data.* OUT: Number of records(rows) present in the table structure.
reproc	Character*(*)	IN: Intent to reprocess the data.

Parameter	Data Type	Definition
start	Integer	IN: Zero-based record location to begin placing the data into the table structure. The *start* location must be contiguous to the location of records already in the table. If *start* = -1 data records will be appended to the end of the table structure. The *start* array must have the same number of elements as the target array has dimensions.
stringlen	Integer	IN/OUT: Minimum length of *fldnm* and *dtype* arrays. Returns the minimum array length actually required to hold the longer of the two strings. It is set to 0 if a functional error occurs.
substr	Character*(*)	OUT: Array of substrings obtained from the *cvalue* array.
tblnm	Character*(*)	IN: ASCII string that will be the name of the table, up to 64 characters long.
tcov	Character*(*)	IN: Description observation period in ECS metadata syntax.
value	<valid type>	IN: Data to store in the in the attribute. If the attribute already exists, the value will be updated. Values should conform to the data types, formats and/or those values enumerated for the attribute in Appendix B, M-API-Supplied Constants. OUT: Value associated with the attribute.

APPENDIX E: ERROR MESSAGES FOR ROUTINES

Table E-1 Error Messages for C Routines

C Routine	Error Message	Description
closeMODISfile	closeMODISfile cannot close a null file.	
	closeMODISfile detected FAIL from HDF function Sdend. Unable to close *filename*.	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	WARNING: closeMODISfile closed new file *filename* without complete header information.	The file has been successfully closed, but completeMODISfile should be used instead so that the required file header information will be included.
completeMODISfile	completeMODISfile unable to continue with empty input.	
	closeMODISfile detected FAIL from HDF function Hclose. Unable to close file.	
	closeMODISfile detected FAIL from HDF function Sdend. Unable to close file.	
	completeMODISfile detected FAIL from HDF procedure Hclose. Unable to close file.	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	WARNING: completeMODISfile revised header data of pre-existing filename file.	The file has been successfully closed, but closeMODISfile should be used instead to prevent modification to the MODIS HDF file's metadata.
	WARNING: completeMODISfile unable to revise header data of filename file open for read-only.	The file has been successfully closed and was accessed only for reading.

C Routine	Error Message	Description
createMODISarray	createMODISarray unable to make a new *arrayname* array with a NULL file MODFILE structure.	
	createMODISarray unable to make a new array without an array name input.	
	createMODISarray unable to make a new *arrayname* array without array dimension input.	
	createMODISarray unable to make a new *arrayname* array without array data type input.	
	createMODISarray unable to make a new *arrayname* array in file opened for read only.	
	createMODISarray found *arrayname* array already exists.	
	createMODISarray found *arrayname* array already exists in data group “*groupname*”.	
	createMODISarray unable to find data group *groupname* to place new *arrayname* array in.	
	createMODISarray unable to create *arrayname* array of data type *data_type.*	
	createMODISarray unable to create *arrayname* array with *rank* dimensions.	
	createMODISarray detected FAIL from HDF procedure SDcreate attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Sdend access while attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Vattach attempting to create *arrayname* array.	
	createMODISarray detected FAIL from HDF procedure Vaddtagref attempting to create *arrayname* array.	

C Routine	Error Message	Description
createMODISarray (continued)	createMODISarray detected FAIL from HDF procedure Vdetach attempting to create *arrayname* array.	
createMODIStable	createMODIStable unable to make a new table without a table name input.	
	createMODIStable unable to make a new *tablename* table with a NULL file MODFILE structure.	
	createMODIStable unable to make a new *tablename* table without field names input.	
	createMODIStable unable to make a new *tablename* table without field data types input.	
	createMODIStable unable to make a new *tablename* table in file opened for read only.	
	createMODIStable found the *tablename* table already exists.	
	createMODISarray found *arrayname* array already exists in data group *"groupname"*.	
	createMODISarray unable to find data group *groupname* to place new *arrayname* array in.	
	createMODIStable unable to create *tablename* table with # byte records.	Vdata table records are limited to 32K each.
	createMODIStable unable to create *tablename* table with *data_type* data types.	
	createMODIStable unable to allocate memory for *fieldname* temporary buffer used to create the *tablename* table.	
	createMODIStable unable to allocate memory for *data_type* temporary buffer used to create *tablename* table.	
	createMODIStable found the *tablename* table to have no fields in the fieldname string *fieldname.*	

C Routine	Error Message	Description
createMODIStable (continued)	createMODIStable unable to support the creation of # fields in the field name string “fieldname” for the “tablename” table.	Vdata table records are limited to fields
	createMODIStable found the *tablename* table to have # data types in the data type string *data_type* instead of #.	One data type must be supplied for each field in the Vdata table.
	createMODIStable detected FAIL from HDF procedure VSattach attempting to create the *tablename* table.	
	createMODIStable detected fail from HDF procedure VSfdefine for *field* and *data_type* of the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure VSsetfields creating the *tablename* table.	
	createMODIStable unable to allocate memory for dummy field buffer used to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure VSwrite creating the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vattach attempting to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vaddtagref attempting to create the *tablename* table.	
	createMODIStable detected FAIL from HDF procedure Vdetach attempting to create the *tablename* table.	
getMODISardims	getMODISardims unable to access the *arrayname* array with a NULL file MODFILE structure.	
	getMODISardims unable to access an array without an array name input.	
	getMODISardims unable to return the *arrayname* array’s dimensions without a dimsizes array.	
	getMODISardims cannot find the *arrayname* array.	

C Routine	Error Message	Description
getMODISardims (continued)	getMODISardims cannot find the *arrayname* array in the *groupname* data group.	
	getMODISardims unable to find the *groupname* data group containing the *arrayname* array.	
	getMODISardims cannot get an sds_id for the *arrayname* array.	
	getMODISardims detected FAIL from HDF procedure SDgetinfo attempting to access the *arrayname* array.	
	getMODISardims detected FAIL from HDF procedure SDendaccess attempting to detach from the *arrayname* array.	The output from getMODISardims may not be valid if SDendaccess fails.
	*rank (if provided) is set to 0 if any of the errors associated with these messages occurs.	
	getMODISardims unable to return the *arrayname* array’s *sds_rank* dimension sizes in a *rank* element dimsizes array.	getMODISardims will not attempt to write to the dimsizes output array, but it will return the rank of the target HDF array structure. The dimsizes array needs to have at least this many elements.
getMODISarinfo	getMODISarinfo unable continue with empty n_elements.	
	getMODISarinfo unable to access an array attribute without an attribute name input.	
	getMODISarinfo unable to access the *attribute* attribute without the name of the array it is associated with.	No *arrayname* attribute was provided.
	getMODISarinfo cannot find array “*arrayname*”.	

C Routine	Error Message	Description
getMODISarinfo (continued)	getMODISarinfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	getMODISarinfo cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.
	getMODISarinfo detected FAIL retrieving the data type string for the *attribute* attribute using DFNT_to_datatype.	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	getMODISarinfo cannot find local array attribute *attribute*.	
	getMODISarinfo detected FAIL from HDF procedure SDselect attempting to read the *attribute* attribute.	
	getMODISarinfo detected FAIL from HDF procedure SDattrinfo attempting to read the *attribute* attribute.	
	getMODISarinfo detected FAIL from HDF procedure SDreadattr attempting to read the *attribute* attribute.	
	getMODISarinfo unable to read local array attribute without output buffer for *attribute*.	
	getMODISarinfo detected FAIL from HDF procedure SDendaccess attempting to read the *attribute* attribute.	**n_elements* is set to 0 if any of the errors associated with the messages above occur.

C Routine	Error Message	Description
getMODISarinfo (continued)	`WARNING: Vgroup groupname contains non-existing SDS object with reference id` *`ref_id`*`.`	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
getMODISarray	`getMODISarray unable to read from the` *`arrayname`* `array with a NULL file MODFILE structure.`	
	`getMODISarray unable to read from an array without an array name input.`	
	`getMODISarray unable to read from the` *`arrayname`* `array without array dimension input.`	
	`getMODISarray unable to read from the` *`arrayname`* `array without a data buffer.`	
	`getMODISarray cannot find the` *`arrayname`* `array.`	
	`getMODISarray cannot find the` *`arrayname`* `array in the` *`groupname`* `data group.`	
	`getMODISarray unable to find the` *`groupname`* `data group containing the` *`arrayname`* `array.`	
	`getMODISarray unable to read data from invalid array structure locations in the` *`arrayname`* `array.`	This error message may be preceeded by one of the following two messages:
	`SDS_footprintOK detected FAIL from HDF procedure Sdgetinfo.`	
	`Unable to access data at invalid array structure locations` *`"start[0] ... start[r]".`*	
	`getMODISarray detected FAIL from HDF procedure SDselect while attempting to read from the` *`arrayname`* `array.`	

C Routine	Error Message	Description
getMODISarray (continued)	getMODISarray detected FAIL from HDF procedure SDgetinfo while attempting to read from the *arrayname* array.	
	getMODISarray detected FAIL from HDF procedure SDwritedata while attempting to read from the *arrayname* array.	
	getMODISarray detected FAIL from HDF procedure SDendaccess while attempting to read from the *arrayname* array.	
getMODISdiminfo	getMODISdiminfo unable continue with empty n_elements.	
	getMODISdiminfo unable to access an array attribute without an attribute name input.	
	getMODISdiminfo unable to access the *attribute* attribute without the name of the array it is associated with.	No *arrayname* attribute was provided.
	getMODISdiminfo cannot find array "*arrayname*".	
	getMODISdiminfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	getMODISdiminfo cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.

C Routine	Error Message	Description
getMODISdiminfo (continued)	`getMODISdiminfo detected FAIL retrieving the data type string for the` *`attribute`* `attribute using DFNT_to_datatype.`	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	`getMODISdiminfo cannot find local array dimension attribute` *`attribute.`*	
	`getMODISdiminfo detected FAIL from HDF procedure SDselect attempting to read the` *`attribute`* `attribute`	
	`getMODISdiminfo detected FAIL from HDF procedure SDgetinfo attempting to read the` *`attribute`* `attribute.`	
	`getMODISdiminfo detected FAIL from HDF procedure SDattrinfo attempting to read the` *`attribute`* `attribute.`	
	`getMODISdiminfo unable to retrieve an` *`attribute`* `attribute for dimension` *`dimension`*`. The` *`arrayname`* `array has` *`rank`* `dimensions.`	
	`getMODISdiminfo detected FAIL from HDF procedure SDgetdimid attempting to read the` *`attribute`* `attribute.`	
	`getMODISdiminfo detected FAIL from HDF procedure SDreadattr attempting to read the` *`attribute`* `attribute.`	
	`getMODISdiminfo unable to read local array attribute without output buffer for` *`attribute`*`.`	
	`getMODISdiminfo detected FAIL from HDF procedure SDendaccess attempting to read the` *`attribute`* `attribute.`	**n_elements* is set to 0 if any of the errors associated with the messages above occur.

C Routine	Error Message	Description
getMODISdiminfo (continued)	`WARNING: Vgroup groupname contains non-existing SDS object with reference id` *`ref_id`*`.`	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
getMODISECSinfo	`getMODISECSinfo can not continue without the n_elements input.`	
	`getMODISECSinfo unable to access an ECS metadata without the parameter name input.`	
	`getMODISECSinfo unable to access the` *`parmName`* `metadata without the name of the global attribute it is stored within.`	
	`getMODISECSinfo unable to access the` *`parmName`* `metadata from ECS global attribute` *`PVLAttrName`* `without the data type input.`	
	`getMODISECSinfo detected fails in procedure MPVL2ODL while attempting to retrieve parameter` *`parmName`* `from ECS global attribute` *`PVLAttrName`*`.`	
	`getMODISECSinfo can not find the` *`parmName`* `metadata.`	
	`getMODISECSinfo found the value for parameter` *`parmName`* `is undefined.`	
	`getMODISECSinfo unable to access the` *`parmName`* `metadata without the output data buffer.`	
	`getMODISECSinfo found unknown ODL value type` *`valueNode->item.type`* `for parameter` *`parmName`*`.`	
getMODISfields	`getMODISfields unable to access the` *`tablename`* `table with a NULL file MODFILE structure.`	
	`getMODISfields unable to access a table without a table name input.`	

C Routine	Error Message	Description
`getMODISfields` (continued)	`getMODISfields cannot find` *`tablename`* `table.`	
	`getMODISfields unable to find the` *`groupname`* `data group containing the` *`tablename`* `table.`	This may be preceeded by one of the following two messages:
	`searchMODISgroup fails to search object` *`objectname`* `in group` *`groupname`* `because Vattach fails.`	
	`searchMODISgroup unable to find the specified Vgroup group` *`groupname`*`.`	
	`getMODISfields cannot find the` *`tablename`* `table in the` *`groupname`* `data group.`	This may be preceeded by the following message:
	`searchMODISgroup fails to obtain` *`objectname`*`'s tag and reference number.`	The Vdata table could not be found in the specified Vgroup data group.
	`getMODISfields detected FAIL from HDF procedure VSattach attempting to access the` *`tablename`* `table.`	
	`getMODISfields detected FAIL from HDF procedure VSgetfields.`	A problem occurred with retrieving information about the number and names of the table's fields.
	`getMODISfields detected FAIL retrieving the data type string for the` *`tablename`* `table using Vfdatatypes.`	This error message may be preceeded by one of the following two messages:
	`VFdatatypes detected FAIL from HDF routine Vfnfields.`	
	`VFdatatypes detected unrecognized HDF number type.`	
	M-API currently does not recognize number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).	
	`getMODISfields detected FAIL from HDF procedure VSinquire.`	A problem occurred with retrieving information about the number of records in the table.

C Routine	Error Message	Description
getMODISfields (continued)	getMODISfields detected FAIL from HDF procedure VSinquire.	A problem occurred with retrieving information about the number of records in the table. **stringlen* is set to 0 if any of the errors associated with the messages above occur.
	getMODISfields unable to fit *tablename* table's <string length> byte field names string into output string of unknown length.	The length of the output string *fieldname* was not provided in the parameter *stringlen*.
	getMODISfields unable to fit the *tablename* table's <string length> byte field names into **stringlen* byte output string.	*stringlen* will return the array length required to hold the table's field names.
	getMODISfields unable to fit *tablename* table's data types string into output string of unknown length.	The length of the output string *data_type* was not provided in the parameter *stringlen*.
	getMODISfields unable to fit the *tablename* table's <string length> byte data types into **stringlen* byte output string. VFdatatypes unable to fit data types into output string.	This error message will be preceeded by: **stringlen* will return the array length required to hold the table's data type string. If both the field names and the data types were requested, the larger of the two array lengths is returned.
getMODISfileinfo	getMODISfileinfo detected FAIL from HDF procedure SDattrinfo. getMODISfileinfo detected FAIL from HDF procedure SDreadattr.	
getMODIStable	getMODIStable unable continue without buffer size information.	A location for *buffsize* information was not provided.
	getMODIStable unable to read from the *tablename* table with a NULL file MODFILE structure. getMODIStable unable to read from a table without a table name input.	

C Routine	Error Message	Description
getMODIStable (continued)	getMODIStable unable to read from the *tablename* table without a data buffer.	
	getMODIStable cannot find *tablename* table.	
	getMODIStable unable to find the *groupname* data group containing the *tablename* table.	This may be preceeded by one of the following two messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	getMODIStable cannot find the *tablename* table in the *groupname* data group.	This may be preceeded by the following message:
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	The Vdata table could not be found in the specified Vgroup data group.
	getMODIStable detected FAIL from HDF procedure VSattach attempting to access the *tablename* table.	
	getMODIStable unable to read data from the *tablename* table from invalid table structure record *start.*	
	getMODIStable unable to read data from the *tablename* table from invalid table structure locations.	Either access to some records or one or more fields requested do not exist in the table.
	getMODIStable detected FAIL from HDF procedure VSsetfields attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSsizeof attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSseek attempting to read *tablename* table.	
	getMODIStable detected FAIL from HDF procedure VSread attempting to read *tablename* table.	**buffsize* is set to 0 if any of the errors associated with the messages above occurs.

C Routine	Error Message	Description
getMODIStable (continued)	getMODIStable detected FAIL from HDF procedure VSinquire.	Should this error occur, getMODIStable will still return MAPIOK (because the data were successfully retrieved) and **buffsize* is set correctly.
	getMODIStable unable to fit <output size> bytes of *tablename* table's data into a *buffsize* byte output buffer.	getMODIStable will not attempt to write to the *data* output buffer, but it will return the buffer length (in bytes) required to hold the requested records from the table.
	WARNING: Vgroup groupname contains non-exist Vdata object with reference id *ref_id*.	Information about a Vdata table that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified Vdata table, it does identify a probable defect in the HDF file.
	WARNING: getMODIStable retrieved dummy record from empty table *tablename.*	The record retrieved from the table does not contain geophysical data. getMODIStable returns MAPIOK (0), however. This situation can only occur if NO geophysical data were written into the table or the single record in the Vdata was not written using M-API.
openMODISfile	openMODISfile unable to access a file without a filename input. openMODISfile unable to open file *filename* without access mode input. openMODISfile unable to allocate memory for a MODIS file structure for file *filename.*	

C Routine	Error Message	Description
openMODISfile (continued)	openMODISfile unable to recognize access type *access* to open file *filename*.	
	openMODISfile unable to find file *filename.*	
	openMODISfile detected FAIL from HDF procedure SDstart opening file *filename*.	May be unable to open the HDF file because it is write-protected.
	openMODISfile detected NULL from HDF function SDIhandle_from_id accessing file *filename*.	
	openMODISfile unable to allocate memory for the MODIS filename *filename*.	
putMODISarinfo	putMODISarinfo unable to write an array attribute without an attribute name input.	
	putMODISarinfo unable to write the *attribute* array attribute without data type information.	
	putMODISarinfo unable to write the *attribute* array attribute without the value buffer.	
	putMODISarinfo unable to write the *attribute* array attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	putMODISarinfo unable to write *n_elements attribute* array attribute values.	
	putMODISarinfo unable to write the *attribute* array attribute in a file opened for read only.	
	putMODISarinfo cannot find array *arrayname*.	

C Routine	Error Message	Description
putMODISarinfo (continued)	putMODISarinfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	putMODISarinfo cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	putMODISarinfo unable to write the *attribute* array attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.
	putMODISarinfo unable to write the *attribute* array attribute of data type *data_type*.	
	putMODISarinfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* array attribute.	
	putMODISarinfo detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* array attribute.	
	putMODISarinfo detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* array attribute	
	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local array attribute, it does identify a probable defect in the HDF file.

C Routine	Error Message	Description
putMODISarray	putMODISarray unable to write to the *arrayname* array with a NULL file MODFILE structure.	
	putMODISarray unable to write to an array without an array name input.	
	putMODISarray unable to write to the *arrayname* array without array dimension input.	
	putMODISarray unable to write to the *arrayname* array without a data buffer.	
	putMODISarray unable to write to the *arrayname* array in file opened for read only.	
	putMODISarray cannot find the *arrayname* array.	
	putMODISarray cannot find the *arrayname* array in the *groupname* data group.	
	putMODISarray unable to find the *groupname* data group containing the *arrayname* array.	
	putMODISarray unable to write data to invalid array structure locations in the *arrayname* array.	This error message may be preceeded by one of the following two messages:
	SDS_footprintOK detected FAIL from HDF procedure SDgetinfo.	
	Unable to access data at invalid array structure locations *"start[0] ... start[r]".*	
	putMODISarray detected FAIL from HDF procedure SDselect while attempting to write to the *arrayname* array.	
	putMODISarray detected FAIL from HDF procedure SDgetinfo while attempting to write to the *arrayname* array.	
	putMODISarray detected FAIL from HDF procedure SDwritedata while attempting to write to the *arrayname* array.	

C Routine	Error Message	Description
putMODISarray (continued)	putMODISarray detected FAIL from HDF procedure SDendaccess while attempting to write to the *arrayname* array.	
putMODISdiminfo	putMODISdiminfo unable to write an dimension attribute without an attribute name input.	
	putMODISdiminfo unable to write the *attribute* dimension attribute without data type information.	
	putMODISdiminfo unable to write the *attribute* dimension attribute without the value buffer.	
	putMODISdiminfo unable to write the *attribute* dimension attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	putMODISdiminfo unable to write *n_elements attribute* dimension attribute values.	
	putMODISdiminfo unable to write the *attribute* dimension attribute in a file opened for read only.	
	putMODISdiminfo cannot find array *arrayname*.	
	putMODISdiminfo unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	putMODISdiminfo cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	putMODISdiminfo unable to write the *attribute* dimension attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.

C Routine	Error Message	Description
putMODISdiminfo (continued)	putMODISdiminfo unable to write the *attribute* dimension attribute of data type *data_type*.	
	putMODISdiminfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDgetinfo attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo unable to write the *attribute* attribute to non-existing dimension *dimension* of the *arrayname* array.	
	putMODISdiminfo detected FAIL from HDF procedure SDgetdimid attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* dimension attribute.	
	putMODISdiminfo detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* dimension attribute.	
	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local dimension attribute, it does identify a probable defect in the HDF file.

C Routine	Error Message	Description
putMODISfileinfo	putMODISfileinfo unable continue with empty input.	
	putMODISfileinfo unable to store *n_elements attribute* global attribute values.	
	putMODISfileinfo unable to write metadata in file opened for read only.	
	putMODISfileinfo unable to identify data type "*data*_type".	
	putMODISfileinfo unable to write *attribute* metadata with a *size* byte value.	
	putMODISfileinfo detected FAIL from HDF procedure SDsetattr.	
putMODIStable	putMODIStable unable to write to the *tablename* table with a NULL file MODFILE structure.	
	putMODIStable unable to write to a table without an table name input.	
	putMODIStable unable to write to the *tablename* table without table dimension input.	
	putMODIStable unable to write to the *tablename* table without a data buffer.	
	putMODIStable unable to write to the *tablename* table in file opened for read only.	
	putMODIStable cannot find the *tablename* table.	
	putMODIStable cannot find the *tablename* table in the *groupname* data group.	
	putMODIStable unable to find the *groupname* data group containing the *tablename* table.	
	putMODIStable detected FAIL from HDF procedure Vattach while attempting to write to the *tablename* table.	

C Routine	Error Message	Description
`putMODIStable` (continued)	`putMODIStable detected FAIL from HDF procedure VSattach while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable detected FAIL from HDF procedure VSinquire while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable unable to place` *`datasize`* `bytes of data into` *`recno record size`* `byte records in` *`tablename`* `table.`	
	`putMODIStable unable to write data to table` *`tablename`* `to invalid table structure record` *`start.`*	The *start* location must be contiguous to the location of records already in the table.
	`putMODIStable detected FAIL from HDF procedure VSseek while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable detected FAIL from HDF procedure VSwrite while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable detected FAIL from M-API procedure set_Vhasdata while attempting to write to the` *`tablename`* `table.`	The first record has successfully been written to the table, however M-API was unable to write an associated attribute into the file. This will cause a subsequent write to the table <u>appending</u> additional records to inadvertantly overwrite this first one.
	Sometimes it is necessary to read from the table structure before writing to it. The following two error messages may occur only in these circumstances:	
	`putMODIStable memory allocation failure while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable detected FAIL from HDF procedure VSread while attempting to write to the` *`tablename`* `table.`	
	`putMODIStable memory allocation failure while attempting to write to the` *`tablename`* `table.`	

C Routine	Error Message	Description
putMODIStable (continued)	putMODIStable detected FAIL from HDF procedure VSread while attempting to write to the *tablename* table.	
substrMODISECSinfo	substrMODISECSinfo unable to continue without char_value input. substrMODISECSinfo unable to continue without n_strings input. substrMODISECSinfo unable to continue without substr pointer array. substrMODISECSinfo unable to continue with invalid *n_elements* n_elements. substrMODISECSinfo unable to fit *loc_n_strings* substrings into **n_strings* pointers *substr* array. substrMODISECSinfo detected MFAIL from MAPI procedure parse_string attempting to parse the *char_value* char_value.	

Table E-2 Error Messages for FORTRAN Routines

FORTRAN Routine	Error Message	Description
CLMFIL	`CLMFIL cannot close a null file.`	
	`CLMFIL detected FAIL from HDF function SDend. Unable to close` *filename*`.`	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	`WARNING: CLMFIL closed new file` *filename* `without complete header information.`	The file has been successfully closed, but CPMFIL should be used instead so that the required file header information will be included.
	`CLMFIL cannot close a non-existing file.`	The *modfil* array does not contain valid file access information.
CPMFIL	`CPMFIL unable to continue with empty input.`	
	`CLMFIL detected FAIL from HDF function Hclose. Unable to close file.`	
	`CLMFIL detected FAIL from HDF function Sdend. Unable to close file.`	
	`CPMFIL detected FAIL from HDF procedure Hclose. Unable to close file.`	File could not be closed because access identifiers to data objects are still attached to the file. Changes to the file may be lost.
	`WARNING: CPMFIL revised header data of pre-existing filename file.`	The file has been successfully closed, but CLMFIL should be used instead to prevent modification to the MODIS HDF file's metadata.
	`WARNING: CPMFIL unable to revise header data of filename file open for read-only.`	The file has been successfully closed and was accessed only for reading.
CRMAR	`CRMAR unable to make a new` *arrayname* `array with a NULL file MODFILE structure.`	
	`CRMAR unable to make a new array without an array name input.`	

FORTRAN Routine	Error Message	Description
CRMAR (cont)	CRMAR unable to make a new *arrayname* array without array dimension input.	
	CRMAR unable to make a new *arrayname* array without array data type input.	
	CRMAR unable to make a new *arrayname* array in file opened for read only.	
	CRMAR found *arrayname* array already exists.	
	CRMAR found *arrayname* array already exists in data group "*groupname*".	
	CRMAR unable to find data group *groupname* to place new *arrayname* array in.	
	CRMAR unable to create *arrayname* array of data type *data_type.*	
	CRMAR unable to create *arrayname* array with *rank* dimensions.	
	CRMAR detected FAIL from HDF procedure SDcreate attempting to create *arrayname* array.	
	CRMAR detected FAIL from HDF procedure Sdend access while attempting to create *arrayname* array.	
	CRMAR detected FAIL from HDF procedure Vattach attempting to create *arrayname* array.	
	CRMAR detected FAIL from HDF procedure Vaddtagref attempting to create *arrayname* array.	
	CRMAR detected FAIL from HDF procedure Vdetach attempting to create *arrayname* array.	
CRMTBL	CRMTBL unable to make a new table without a table name input.	
	CRMTBL unable to make a new *tablename* table with a NULL file MODFILE structure.	
	CRMTBL unable to make a new *tablename* table without field names input.	
	CRMTBL unable to make a new *tablename* table without field data types input.	
	CRMTBL unable to make a new *tablename* table in file opened for read only.	
	CRMTBL found the *tablename* table already exists.	

FORTRAN Routine	Error Message	Description
CRMTBL (cont)	CRMAR found *arrayname* array already exists in data group "*groupname*".	
	CRMAR unable to find data group *groupname* to place new *arrayname* array in.	
	CRMTBL unable to create *tablename* table with # byte records.	Vdata table records are limited to 32K each.
	CRMTBL unable to create *tablename* table with *data_type* data types.	
	CRMTBL unable to allocate memory for *fieldname* temporary buffer used to create the *tablename* table.	
	CRMTBL unable to allocate memory for *data_type* temporary buffer used to create *tablename* table.	
	CRMTBL found the *tablename* table to have no fields in the fieldname string *fieldname.*	
	CRMTBL unable to support the creation of # fields in the field name string "fieldname" for the "tablename" table.	Vdata table records are limited to fields
	CRMTBL found the *tablename* table to have # data types in the data type string *data_type* instead of #.	One data type must be supplied for each field in the Vdata table.
	CRMTBL detected FAIL from HDF procedure VSattach attempting to create the *tablename* table.	
	CRMTBL detected fail from HDF procedure VSfdefine for *field* and *data_type* of the *tablename* table.	
	CRMTBL detected FAIL from HDF procedure VSsetfields creating the *tablename* table.	
	CRMTBL unable to allocate memory for dummy field buffer used to create the *tablename* table.	
	CRMTBL detected FAIL from HDF procedure VSwrite creating the *tablename* table.	
	CRMTBL detected FAIL from HDF procedure Vattach attempting to create the *tablename* table.	
	CRMTBL detected FAIL from HDF procedure Vaddtagref attempting to create the *tablename* table.	

FORTRAN Routine	Error Message	Description
CRMTBL (cont)	CRMTBL detected FAIL from HDF procedure Vdetach attempting to create the *tablename* table. nccrmtbl failed at data_type conversion. nccrmtbl out of memory.	
GMAR	GMAR unable to read from the *arrayname* array with a NULL file MODFILE structure. GMAR unable to read from an array without an array name input. GMAR unable to read from the *arrayname* array without array dimension input. GMAR unable to read from the *arrayname* array without a data buffer. GMAR cannot find the *arrayname* array. GMAR cannot find the *arrayname* array in the *groupname* data group. GMAR unable to find the *groupname* data group containing the *arrayname* array.	
	GMAR unable to read data from invalid array structure locations in the *arrayname* array. SDS_footprintOK detected FAIL from HDF procedure Sdgetinfo. Unable to access data at invalid array structure locations *"start[0] ... start[r]".*	This error message may be preceeded by one of the following two messages:
	GMAR detected FAIL from HDF procedure SDselect while attempting to read from the *arrayname* array. GMAR detected FAIL from HDF procedure SDgetinfo while attempting to read from the *arrayname* array. GMAR detected FAIL from HDF procedure SDwritedata while attempting to read from the *arrayname* array. GMAR detected FAIL from HDF procedure SDendaccess while attempting to read from the *arrayname* array.	

FORTRAN Routine	Error Message	Description
GMARDM	GMARDM detected FAIL from HDF procedure SDendaccess attempting to detach from the *arrayname* array.	The output from GMARDM may not be valid if SDendaccess fails. *rank (if provided) is set to 0 if any of the errors associated with these messages occurs.
	GMARDM unable to return the *arrayname* array's *sds_rank* dimension sizes in a *rank* element dimsizes array.	GMARDM will not attempt to write to the dimsizes output array, but it will return the rank of the target HDF array structure. The dimsizes array needs to have at least this many elements.
	GMARDM unable to access the *arrayname* array with a NULL file MODFILE structure.	
	GMARDM unable to access an array without an array name input.	
	GMARDM unable to return the *arrayname* array's dimensions without a dimsizes array.	
	GMARDM cannot find the *arrayname* array.	
	GMARDM cannot find the *arrayname* array in the *groupname* data group.	
	GMARDM unable to find the *groupname* data group containing the *arrayname* array.	
	GMARDM cannot get an sds_id for the *arrayname* array.	
	GMARDM detected FAIL from HDF procedure SDgetinfo attempting to access the *arrayname* array.	
	GMARDM detected FAIL from HDF procedure SDendaccess attempting to detach from the *arrayname* array.	The output from GMARDM may not be valid if SDendaccess fails. *rank (if provided) is set to 0 if any of the errors associated with these messages occurs.

FORTRAN Routine	Error Message	Description
GMARDM (cont)	GMARDM unable to return the *arrayname* array's *sds_rank* dimension sizes in a *rank* element dimsizes array.	GMARDM will not attempt to write to the dimsizes output array, but it will return the rank of the target HDF array structure. The dimsizes array needs to have at least this many elements.
GMARIN	GMARIN unable continue with empty n_elements.	
	GMARIN unable to access an array attribute without an attribute name input.	No *arrayname* attribute was provided.
	GMARIN unable to access the *attribute* attribute without the name of the array it is associated with.	
	GMARIN cannot find array "*arrayname*".	
	GMARIN unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	GMARIN cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.
	GMARIN detected FAIL retrieving the data type string for the *attribute* attribute using DFNT_to_datatype.	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	GMARIN cannot find local array attribute *attribute*.	
	GMARIN detected FAIL from HDF procedure SDselect attempting to read the *attribute* attribute.	

FORTRAN Routine	Error Message	Description
GMARIN (cont)	`GMARIN detected FAIL from HDF procedure SDattrinfo attempting to read the` *attribute* `attribute.`	
	`GMARIN detected FAIL from HDF procedure SDreadattr attempting to read the` *attribute* `attribute.`	
	`GMARIN unable to read local array attribute without output buffer for` *attribute*`.`	
	`GMARIN detected FAIL from HDF procedure SDendaccess attempting to read the` *attribute* `attribute.`	**n_elements* is set to 0 if any of the errors associated with the messages above occur.
	`WARNING: Vgroup groupname contains non-existing SDS object with reference id` *ref_id*`.`	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
	`GMARIN unable to update the data type because the memory for dtype is too small.`	
	`GMARIN unable continue with empty n_elements.`	
	`GMARIN unable to access an array attribute without an attribute name input.`	
	`GMARIN unable to access the` *attribute* `attribute without the name of the array it is associated with.`	No *arrayname* attribute was provided.
	`GMARIN cannot find array “`*arrayname*`”.`	
	`GMARIN unable to find the` *groupname* `data group containing the` *arrayname* `array.`	This may be preceeded by one of the following three messages:
	`searchMODISgroup fails to search object` *objectname* `in group` *groupname* `because Vattach fails.`	
	`searchMODISgroup unable to find the specified Vgroup group` *groupname*`.`	
	`searchMODISgroup fails to obtain` *objectname*`'s tag and reference number.`	

FORTRAN Routine	Error Message	Description
GMARIN (cont)	`GMARIN cannot find the` *`arrayname`* `array in the` *`groupname`* `data group.`	The Vdata table could not be found in the specified Vgroup data group.
	`GMARIN detected FAIL retrieving the data type string for the` *`attribute`* `attribute using DFNT_to_datatype.`	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	`GMARIN cannot find local array attribute` *`attribute`*`.`	
	`GMARIN detected FAIL from HDF procedure SDselect attempting to read the` *`attribute`* `attribute.`	
	`GMARIN detected FAIL from HDF procedure SDattrinfo attempting to read the` *`attribute`* `attribute.`	
	`GMARIN detected FAIL from HDF procedure SDreadattr attempting to read the` *`attribute`* `attribute.`	
	`GMARIN unable to read local array attribute without output buffer for` *`attribute`*`.`	
	`GMARIN detected FAIL from HDF procedure SDendaccess attempting to read the` *`attribute`* `attribute.`	**n_elements* is set to 0 if any of the errors associated with the messages above occur.
	`WARNING: Vgroup groupname contains non-existing SDS object with reference id` *`ref_id`*`.`	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
	`GMARIN unable to update the data type because the memory for dtype is too small.`	

FORTRAN Routine	Error Message	Description
GMDMIN	GMDMIN unable continue with empty n_elements.	
	GMDMIN unable to access an array attribute without an attribute name input.	
	GMDMIN unable to access the *attribute* attribute without the name of the array it is associated with.	No *arrayname* attribute was provided.
	GMDMIN cannot find array "*arrayname*".	
	GMDMIN unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	GMDMIN cannot find the *arrayname* array in the *groupname* data group.	The Vdata table could not be found in the specified Vgroup data group.
	GMDMIN detected FAIL retrieving the data type string for the *attribute* attribute using DFNT_to_datatype.	M-API currently does not recogniize the HDF number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	GMDMIN cannot find local array dimension attribute *attribute.*	
	GMDMIN detected FAIL from HDF procedure SDselect attempting to read the *attribute* attribute	
	GMDMIN detected FAIL from HDF procedure SDgetinfo attempting to read the *attribute* attribute.	
	GMDMIN detected FAIL from HDF procedure SDattrinfo attempting to read the *attribute* attribute.	

FORTRAN Routine	Error Message	Description
GMDMIN (cont)	GMDMIN unable to retrieve an *attribute* attribute for dimension *dimension*. The *arrayname* array has *rank* dimensions.	
	GMDMIN detected FAIL from HDF procedure SDgetdimid attempting to read the *attribute* attribute.	
	GMDMIN detected FAIL from HDF procedure SDreadattr attempting to read the *attribute* attribute.	
	GMDMIN unable to read local array attribute without output buffer for *attribute*.	
	GMDMIN detected FAIL from HDF procedure SDendaccess attempting to read the *attribute* attribute.	**n_elements* is set to 0 if any of the errors associated with the messages above occur.
	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified local array attribute, it does identify a probable defect in the HDF file.
	GMDMIN unable to update the data type because the memory for dtype is too small.	
GMFIN	GMFIN detected FAIL from HDF procedure SDattrinfo.	
	GMFIN detected FAIL from HDF procedure SDreadattr.	
GMFLDS	GMFLDS unable to access the *tablename* table with a NULL file MODFILE structure.	
	GMFLDS unable to access a table without a table name input.	
	GMFLDS cannot find *tablename* table.	
	GMFLDS unable to find the *groupname* data group containing the *tablename* table.	This may be preceeded by one of the following two messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	

FORTRAN Routine	Error Message	Description
GMFLDS (cont)	GMFLDS cannot find the *tablename* table in the *groupname* data group.	This may be preceeded by the following message:
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	The Vdata table could not be found in the specified Vgroup data group.
	GMFLDS detected FAIL from HDF procedure VSattach attempting to access the *tablename* table.	A problem occurred with retrieving information about the number and names of the table’s fields.
	GMFLDS detected FAIL from HDF procedure VSgetfields.	
	GMFLDS detected FAIL retrieving the data type string for the *tablename* table using Vfdatatypes.	This error message may be preceeded by one of the following two messages:
	VFdatatypes detected FAIL from HDF routine Vfnfields. VFdatatypes detected unrecognized HDF number type number type.	M-API currently does not recogniize number types 3 (unsigned char), 7 (float128), 27 (unsigned int64), 28 (int128), 30 (unsigned int128), 42 (char16), 43 (unsigned char 16), or any greater than 512 (machine specific, custom, or little endian storage formats).
	GMFLDS detected FAIL from HDF procedure VSinquire.	A problem occurred with retrieving information about the number of records in the table.
	GMFLDS detected FAIL from HDF procedure VSinquire.	A problem occurred with retrieving information about the number of records in the table. **stringlen* is set to 0 if any of the errors associated with the messages above occur.
	GMFLDS unable to fit *tablename* table's <string length> byte field names string into output string of unknown length.	The length of the output string *fieldname* was not provided in the parameter *stringlen*.

FORTRAN Routine	Error Message	Description
GMFLDS (cont)	GMFLDS unable to fit the *tablename* table's <string length> byte field names into **stringlen* byte output string.	*stringlen* will return the array length required to hold the table’s field names.
	GMFLDS unable to fit *tablename* table's data types string into output string of unknown length.	The length of the output string *data_type* was not provided in the parameter *stringlen*.
	GMFLDS unable to fit the *tablename* table's <string length> byte data types into **stringlen* byte output string. VFdatatypes unable to fit data types into output string.	This error message will be preceeded by: **stringlen* will return the array length required to hold the table’s data type string. If both the field names and the data types were requested, the larger of the two array lengths is returned.
GMTBL	GMTBL unable continue without buffer size information.	A location for *buffsize* information was not provided.
	GMTBL unable to read from the *tablename* table with a NULL file MODFILE structure.	
	GMTBL unable to read from a table without a table name input.	
	GMTBL unable to read from the *tablename* table without a data buffer.	
	GMTBL cannot find *tablename* table.	
	GMTBL unable to find the *groupname* data group containing the *tablename* table. searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails. searchMODISgroup unable to find the specified Vgroup group *groupname*.	This may be preceeded by one of the following two messages: The Vdata table could not be found in the specified Vgroup data group.
	GMTBL cannot find the *tablename* table in the *groupname* data group. searchMODISgroup fails to obtain *objectname*'s tag and reference number.	This may be preceeded by the following message:
	GMTBL detected FAIL from HDF procedure VSattach attempting to access the *tablename* table.	

FORTRAN Routine	Error Message	Description
GMTBL (cont)	`GMTBL unable to read data from the` *`tablename`* `table from invalid table structure record` *`start.`*	
	`GMTBL unable to read data from the` *`tablename`* `table from invalid table structure locations.`	Either access to some records or one or more fields requested do not exist in the table.
	`GMTBL detected FAIL from HDF procedure VSsetfields attempting to read` *`tablename`* `table.`	
	`GMTBL detected FAIL from HDF procedure VSsizeof attempting to read` *`tablename`* `table.`	
	`GMTBL detected FAIL from HDF procedure VSseek attempting to read` *`tablename`* `table.`	
	`GMTBL detected FAIL from HDF procedure VSread attempting to read` *`tablename`* `table.`	**buffsize* is set to 0 if any of the errors associated with the messages above occurs.
	`GMTBL detected FAIL from HDF procedure VSinquire.`	Should this error occur, getMODIStable will still return MAPIOK (because the data were successfully retrieved) and **buffsize* is set correctly.
	`GMTBL unable to fit <output size> bytes of` *`tablename`* `table's data into a` *`buffsize`* `byte output buffer.`	getMODIStable will not attempt to write to the *data* output buffer, but it will return the buffer length (in bytes) required to hold the requested records from the table.
	`WARNING: Vgroup groupname contains non-exist Vdata object with reference id` *`ref_id`*`.`	Information about a Vdata table that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent reading the specified Vdata table, it does identify a probable defect in the HDF file.

FORTRAN Routine	Error Message	Description
GMTBL (cont)	`WARNING: getMODIStable retrieved dummy record from empty table` *`tablename.`*	The record retrieved from the table does not contain geophysical data. getMODIStable returns MAPIOK (0), however. This situation can only occur if NO geophysical data were written into the table or the single record in the Vdata was not written using M-API.

OPMFIL	`OPMFIL unable to access a file without a filename input.`
	
	`OPMFIL unable to open file` *`filename`* `without access mode input.`
	
	`OPMFIL unable to allocate memory for a MODIS file structure for file` *`filename.`*
	
	`OPMFIL unable to recognize access type` *`access`* `to open file` *`filename`*`.`
	
	`OPMFIL unable to find file` *`filename.`*
	
	`OPMFIL detected FAIL from HDF procedure SDstart opening file` *`filename`*`.`
	
	`OPMFIL detected NULL from HDF function SDIhandle_from_id accessing file` *`filename`*`.`
	
	`openMODISfile unable to allocate memory for the MODIS filename` *`filename`*`.`
		May be unable to open the HDF file because it is write-protected.

PMAR	`PMAR unable to write to the` *`arrayname`* `array with a NULL file MODFILE structure.`
	
	`PMAR unable to write to an array without an array name input.`
	
	`PMAR unable to write to the` *`arrayname`* `array without array dimension input.`
	
	`PMAR unable to write to the` *`arrayname`* `array without a data buffer.`
	
	`PMAR unable to write to the` *`arrayname`* `array in file opened for read only.`
	
	`PMAR cannot find the` *`arrayname`* `array.`
	
	`PMAR cannot find the` *`arrayname`* `array in the` *`groupname`* `data group.`
	
	`PMAR unable to find the` *`groupname`* `data group containing the` *`arrayname`* `array.`	

FORTRAN Routine	**Error Message**	**Description**
PMAR (cont)	PMAR unable to write data to invalid array structure locations in the *arrayname* array.	
	SDS_footprintOK detected FAIL from HDF procedure Sdgetinfo.	This error message may be preceeded by one of the following two messages:
	Unable to access data at invalid array structure locations *“start[0] ... start[r]”.*	
	PMAR detected FAIL from HDF procedure SDselect while attempting to write to the *arrayname* array.	
	PMAR detected FAIL from HDF procedure SDgetinfo while attempting to write to the *arrayname* array.	
	PMAR detected FAIL from HDF procedure SDwritedata while attempting to write to the *arrayname* array.	
	PMAR detected FAIL from HDF procedure SDendaccess while attempting to write to the *arrayname* array.	
PMARIN	PMARIN unable to write an array attribute without an attribute name input.	
	PMARIN unable to write the *attribute* array attribute without data type information.	
	PMARIN unable to write the *attribute* array attribute without the value buffer.	
	PMARIN unable to write the *attribute* array attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	PMARIN unable to write *n_elements attribute* array attribute values.	
	PMARIN unable to write the *attribute* array attribute in a file opened for read only.	
	PMARIN cannot find array *arrayname*.	
	PMARIN unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	

FORTRAN Routine	Error Message	Description
PMARIN (cont)	PMARIN cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	PMARIN unable to write the *attribute* array attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.
	PMARIN unable to write the *attribute* array attribute of data type *data_type*.	
	PMARIN detected FAIL from HDF procedure SDselect attempting to write the *attribute* array attribute.	
	PMARIN detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* array attribute.	
	PMARIN detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* array attribute	
	WARNING: Vgroup groupname contains non-existing SDS object with reference id *ref_id*.	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local array attribute, it does identify a probable defect in the HDF file.
PMDMIN	PMDMIN unable to write an dimension attribute without an attribute name input.	
	PMDMIN unable to write the *attribute* dimension attribute without data type information.	
	PMDMIN unable to write the *attribute* dimension attribute without the value buffer.	
	PMDMIN unable to write the *attribute* dimension attribute without the name of the array it is associated with.	No *arrayname* argument was provided.
	PMDMIN unable to write *n_elements attribute* dimension attribute values.	
	PMDMIN unable to write the *attribute* dimension attribute in a file opened for read only.	
	PMDMIN cannot find array *arrayname*.	

FORTRAN Routine	**Error Message**	**Description**
PMDMIN (cont)	PMDMIN unable to find the *groupname* data group containing the *arrayname* array.	This may be preceeded by one of the following three messages:
	searchMODISgroup fails to search object *objectname* in group *groupname* because Vattach fails.	
	searchMODISgroup unable to find the specified Vgroup group *groupname*.	
	searchMODISgroup fails to obtain *objectname*'s tag and reference number.	
	PMDMIN cannot find the *arrayname* array in the *groupname* data group.	The SDS array structure could not be found in the specified Vgroup data group.
	PMDMIN unable to write the *attribute* dimension attribute with a *size* byte value.	Each HDF attribute is limited to 32K of memory.
	PMDMIN unable to write the *attribute* dimension attribute of data type *data_type*.	
	PMDMIN detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	
	PMDMIN detected FAIL from HDF procedure SDgetinfo attempting to write the *attribute* dimension attribute.	
	PMDMIN detected FAIL from HDF procedure SDselect attempting to write the *attribute* dimension attribute.	
	PMDMIN unable to write the *attribute* attribute to non-existing dimension *dimension* of the *arrayname* array.	
	PMDMIN detected FAIL from HDF procedure SDgetdimid attempting to write the *attribute* dimension attribute.	
	PMDMIN detected FAIL from HDF procedure SDsetattr attempting to write the *attribute* dimension attribute.	
	PMDMIN detected FAIL from HDF procedure SDendaccess attempting to write the *attribute* dimension attribute.	

FORTRAN Routine	Error Message	Description
PMDMIN (cont)	`WARNING: Vgroup groupname contains non-existing SDS object with reference id` *`ref_id`*`.`	Information about an SDS array structure that doesn't really exist has been found in the Vgroup data group being accessed. While this will not directly prevent writing the specified local dimension attribute, it does identify a probable defect in the HDF file.
PMFIN	`PMFIN unable continue with empty input.`	
	`PMFIN unable to store` *`n_elements attribute`* `global attribute values.`	
	`PMFIN unable to write metadata in file opened for read only.`	
	`PMFIN unable to identify data type “`*`data_`*`type”.`	
	`PMFIN unable to write` *`attribute`* `metadata with a` *`size`* `byte value.`	
	`PMFIN detected FAIL from HDF procedure SDsetattr.`	
PMTBL	`PMTBL unable to write to the` *`tablename`* `table with a NULL file MODFILE structure.`	
	`PMTBL unable to write to a table without an table name input.`	
	`PMTBL unable to write to the` *`tablename`* `table without table dimension input.`	
	`PMTBL unable to write to the` *`tablename`* `table without a data buffer.`	
	`PMTBL unable to write to the` *`tablename`* `table in file opened for read only.`	
	`PMTBL cannot find the` *`tablename`* `table.`	
	`PMTBL cannot find the` *`tablename`* `table in the` *`groupname`* `data group.`	
	`PMTBL unable to find the` *`groupname`* `data group containing the` *`tablename`* `table.`	
	`PMTBL detected FAIL from HDF procedure Vattach while attempting to write to the` *`tablename`* `table.`	
	`PMTBL detected FAIL from HDF procedure VSattach while attempting to write to the` *`tablename`* `table.`	